



MGT.0000042645

BÙI VIỆT HÀ

PYTHON CƠ BẢN?

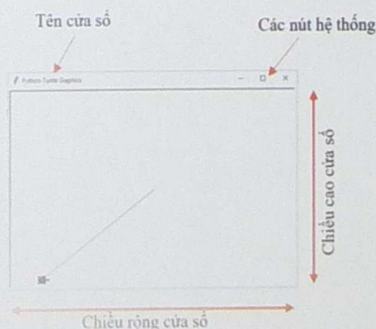
(Tái bản lần 2)

Hàm kiểm tra năm nhuận có thể viết như sau:

```
1. def nhuan(n):
2.     if n % 400 == 0 or (n % 4 == 0 and n % 100 != 0):
3.         return True
4.     else:
5.         return False
```

Kết quả của hàm cho một số giá trị năm:

```
>>> nhuan(2000)
True
>>> nhuan(2100)
False
>>>
```



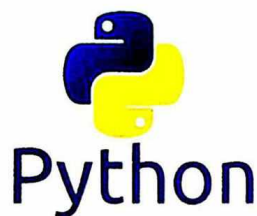
Vị trí của cửa sổ đồ họa Turtle trên màn hình máy tính.



& QTKD
VIỆN



NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI



Python là ngôn ngữ lập trình bậc cao đang phát triển rất nhanh và sẽ được đưa vào nhà trường phổ thông và đại học của Việt Nam như một chương trình bắt buộc. Đây là cuốn sách đầu tiên, Python CƠ BẢN, dành cho người mới bắt đầu học ngôn ngữ lập trình này.

Sách bao gồm 16 chương với trên 350 bài tập từ đơn giản đến nâng cao, phù hợp cho mọi đối tượng từ cấp THCS, THPT đến sinh viên đại học. Sách cũng có thể dùng cho giáo viên dạy Tin học các trường phổ thông và đại học. Mỗi chương sẽ bắt đầu bằng mô tả mục đích, tiếp theo là dãy các hoạt động kiến thức cần học và dạy. Sách có thể dùng cho việc tự học hoặc giáo viên giảng dạy trên lớp. Sau mỗi chương là phần câu hỏi, bài tập chi tiết.

Các chủ đề chính của sách:

1. Bắt đầu với Python.
2. Làm quen môi trường lập trình Python.
3. Input và chuyển đổi dữ liệu.
4. Hàm số.
5. Đối tượng trong Python.
6. Kiểu dữ liệu List. Mảng một chiều.
7. List của List. Mảng nhiều chiều.
8. Khái niệm Module.
9. Xâu ký tự.
10. Đọc và ghi tệp.
11. Câu lệnh điều kiện.
12. Đệ quy.
13. Kiểu dữ liệu Từ điển.
14. Kiểu dữ liệu Tập hợp.
15. Đồ họa con Rùa.
16. Bắt lỗi và kiểm soát lỗi trong Python.

Bùi Việt Hà

005.13
HA



Python cơ bản



Python



NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

Lời nói đầu

Hiện nay ngôn ngữ lập trình bậc cao Python đang nổi lên như một ngôn ngữ lập trình được sử dụng nhiều nhất trên thế giới. Điều này được giải thích bằng các lý do sau:

- Python là ngôn ngữ lập trình bậc cao khá đơn giản, dễ học, dễ viết.
- Cách viết lệnh của Python khá đặc biệt, sử dụng các dấu cách (viết thụt vào) để mô tả các nhóm (block) lệnh. Đặc điểm này làm cho việc viết lệnh Python gần giống với cách viết, trình bày văn bản hàng ngày. Chính đặc điểm này làm cho ngôn ngữ lập trình Python rất dễ viết, trong sáng, ngày càng phát triển và được đưa vào môi trường giáo dục thay cho các ngôn ngữ truyền thống như Pascal, C hay Java.
- Python là ngôn ngữ mã nguồn mở và cho phép cộng đồng có thể đóng góp bằng cách bổ sung các module, các kho hàm số, thư viện thuật toán. Điều này làm cho Python phát triển bùng nổ trong giới khoa học và giáo dục đại học. Đặc biệt trong một số ngành mũi nhọn của CNTT như IoT, trí tuệ nhân tạo (AI), dữ liệu lớn (big data) và CMCN 4.0, các phát triển rất nhanh thời gian gần đây của công nghệ đều gắn liền với Python.
- Ngoài các lý do nêu trên, Python còn có một tính chất khác biệt nữa: Python là ngôn ngữ thông dịch và luôn có môi trường tương tác Python Shell đi kèm. Chính môi trường tương tác này sẽ giúp ích rất nhiều cho những người muốn làm quen và học Python.

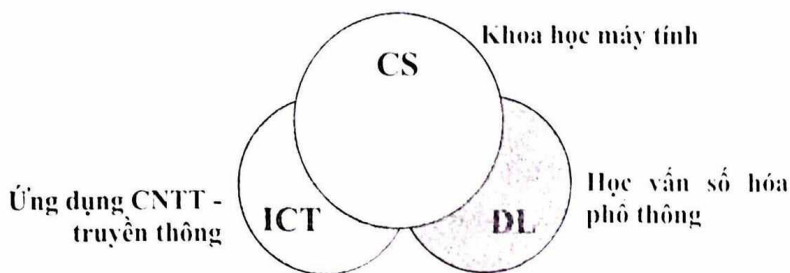
Thời gian gần đây, cùng với phong trào STEM phát triển mạnh, các ngôn ngữ lập trình trực quan lần lượt ra đời như Scratch đã tạo ra một cuộc cách mạng mới, bùng nổ của giáo dục Tin học trong nhà trường. Lần đầu tiên trong lịch sử, định hướng Khoa học máy tính của môn Tin học được đưa xuống cho học sinh Tiểu học. Cuộc cách mạng này sẽ có ý nghĩa to lớn và lâu dài mà chúng ta chưa thể đánh giá hết ngày hôm nay. Rất may mắn là Python chính là ngôn ngữ lập trình thích hợp dùng trong nhà trường, cho lứa tuổi cấp THCS, THPT, phù hợp với cả 2 định hướng phát triển chính của môn Tin học hiện nay là STEM và ALGORITHM. Tôi sẽ phân tích sâu hơn ý nghĩa này của Python.

Như chúng ta đã biết, Bộ Giáo dục và Đào tạo đã ban hành Chương trình Giáo dục Phổ thông mới, còn gọi là Chương trình 2018. Trong chương trình này môn Tin học có nhiều thay đổi lớn nhất: Môn Tin học sẽ là môn học chính thức cho học sinh từ lớp 3 đến lớp 12. Từ cấp THPT, tức là từ lớp 10, môn Tin học sẽ được tách thành 2 định hướng riêng, Tin học ứng dụng (ICT) và Khoa học máy tính (CS).

Một trong những thay đổi lớn nhất của môn Tin học trong Chương trình 2018 là cách tiếp cận mới về nội dung của môn học này. Cách tiếp cận này bắt nguồn từ những phát minh mới nhất thời gian gần đây như CMCN 4.0, các ứng dụng đa dạng của trí tuệ nhân tạo AI, Internet vạn vật IoT, sự xuất hiện của các ngôn ngữ lập trình trực quan như Scratch, sự phát triển bùng nổ của hướng giáo dục STEM trong trường học.

Theo mô hình Chương trình 2018, kiến thức môn Tin học trong trường phổ thông sẽ được phân tách thành 3 mạch (phân môn) chính như sau:

- 1. Khoa học máy tính (CS - Computer Science).**
- 2. CNTT và ứng dụng (IT - Information Technology).**
- 3. Học văn số hóa phổ thông (DL - Digital Literacy).**



Trong mô hình trên, phân môn **Khoa học máy tính (CS)** sẽ là quan trọng nhất, đóng vai trò trung tâm, cốt lõi của môn Tin học trong tương lai.

Trong 3 hướng này, ICT và DL là những nội dung mà giáo viên Tin học hiện nay vẫn đang giảng dạy, do đó sẽ không có gì ngỡ ngàng. Riêng hướng CS là hoàn toàn mới và khác biệt so với hiện nay.

Trong phần ghi Mục tiêu tổng quát của Chương trình môn Tin học mới, bản Đề cương mô tả như sau (xem tài liệu [42]):

Chương trình môn Tin học được xây dựng với mục tiêu góp phần hình thành, phát triển những phẩm chất chủ yếu và năng lực chung, đặc biệt là năng lực tin học đã được xác định trong Chương trình giáo dục phổ thông tổng thể. Môn Tin học trang bị cho học sinh hệ thống kiến thức tin học phổ thông gồm ba mạch kiến thức hoà quyện:

- Học văn số hoá phổ thông nhằm giúp học sinh có khả năng hoà nhập với xã hội hiện đại, sử dụng được các thiết bị số và phần mềm cơ bản thông dụng một cách có đạo đức, văn hoá và tuân thủ pháp luật.
- Công nghệ thông tin và truyền thông nhằm giúp học sinh có khả năng sử dụng và áp dụng hệ thống máy tính giải quyết vấn đề thực tế một cách hiệu quả và sáng tạo.
- Khoa học máy tính nhằm giúp học sinh hiểu biết các nguyên tắc cơ bản và thực tiễn của tư duy máy tính; tạo cơ sở cho việc thiết kế và phát triển các hệ thống máy tính.

Có thể tóm tắt nội dung, bản chất và định hướng phát triển tương lai của 3 nhóm phân môn của môn Tin học trong bảng sau:

DL - Học văn số hóa phổ thông	ICT - Ứng dụng CNTT và truyền thông	CS - Khoa học máy tính
Các kỹ năng cơ bản, tối thiểu cần có trong thời đại số hóa, ví dụ: kỹ năng sử dụng chuột, gõ bàn phím; soạn thảo văn bản, bảng tính, trình chiếu; khai thác Internet; sử dụng thư điện tử và mạng xã hội.	Sử dụng công nghệ xử lý số, phần mềm để ứng dụng và tạo ra các sản phẩm số phục vụ theo yêu cầu. Ví dụ đồ họa, phim, ảnh, ứng dụng trong các công việc đời sống.	Lý thuyết và thực hành về biểu diễn số trong máy tính, xử lý số, tư duy giải quyết vấn đề, thiết lập chương trình, thuật toán, tư duy máy tính.

DL - Học văn số hóa phổ thông	ICT - Ứng dụng CNTT và truyền thông	CS - Khoa học máy tính
Đây là những kỹ năng, năng lực tối thiểu đơn giản nhất cần có của mọi công dân trong thế giới số hiện nay. Có thể ví như học đếm trong môn Toán và học đánh vần trong môn Tiếng Việt.	Mạng kiến thức này sẽ được phân thành 2 nhóm: 1. Nhóm ứng dụng CNTT, phần mềm vào các chuyên ngành, môn học khác với Tin học. 2. Nhóm các ứng dụng CNTT, truyền thông bên trong ICT.	Lỗi của môn học này là khái niệm tư duy máy tính (hay tư duy thuật toán). Đây là kiến thức cơ bản nằm bên dưới các khái niệm quen thuộc như phần mềm, phần cứng máy tính. Đi cùng mô hình tư duy này là một tập hợp các kiến thức lõi, cơ bản về lý thuyết cũng như thực hành, kỹ năng, năng lực phân tích, mô phỏng và giải quyết vấn đề.
Bên cạnh các kỹ năng số cơ bản, các kiến thức về bản quyền trí tuệ, đạo đức, văn hóa, pháp luật khi tham gia mạng xã hội số cũng là một cấu thành quan trọng của nhánh phân môn này.	Nhóm thứ nhất sẽ được chuyển giao cho các bộ môn chuyên biệt. Môn Tin học chỉ còn tập trung ở nhóm thứ hai và sẽ định hướng cho nghề Tin học ứng dụng.	Đây là định hướng chính, cơ bản nhất và quan trọng nhất của môn Tin học trong tương lai. Chính hướng này sẽ quyết định đề môn Tin học trở thành một môn học lõi, có tính khoa học chặt chẽ, có tính kế thừa và phát triển liên tục cho các bậc học tiếp theo.

Như vậy trong môn Tin học mới, khái niệm **Tư duy máy tính** sẽ đóng vai trò trung tâm của môn học này. Chúng ta dành một chút thời gian để tìm hiểu kỹ hơn khái niệm quan trọng này.

Tư duy máy tính: năng lực tư duy quan trọng nhất của Khoa học máy tính hay môn Tin học trong nhà trường.

Xuất xứ của khái niệm **Tư duy máy tính (computational thinking)**.

Người đầu tiên nhắc đến cụm từ này là **Seymour Papert** (29/2/1928 – 31/7/2016), Giáo sư toán Đại học MIT đồng thời là tác giả của phần mềm và ngôn ngữ lập trình LOGO. Ông là người đầu tiên muốn đưa việc giảng dạy thuật toán bằng phần mềm LOGO cho học sinh nhỏ tuổi.

“... the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent”.

Tạm dịch:

“... là các quá trình tư duy bao gồm cả mô tả và lời giải bài toán sao cho lời giải có thể được thể hiện rõ ràng và được thực hiện một cách hiệu quả bởi các tác tử xử lý thông tin”.

Giáo sư **Jeannette Wing** (từng là phó chủ tịch Microsoft) là người đầu tiên đưa ra định nghĩa của khái niệm này (tư duy máy tính), như sau:

"The solution can be carried out by a human or machine, or more generally, by combinations of humans and machines".

Tạm dịch:

"Lời giải phải (và có thể) được thực hiện bởi con người hoặc máy tính, hoặc tổng quát hơn, bởi sự kết hợp (đồng thời) của con người và máy tính".

Như vậy **tư duy máy tính** chính là kỹ năng rất cơ bản để chúng ta có thể biết, hiểu một cách có lý, logic về thế giới xung quanh dựa trên sức mạnh của máy tính.

Phân môn "Khoa học máy tính" là môn học dạy các nguyên tắc lý thuyết và thực hành cho mô hình tính toán máy tính và các ứng dụng của mô hình này. Lời của môn học này là khái niệm **tư duy máy tính** hay **tư duy thuật toán**. Đây là mô hình tư duy lõi, cơ bản nằm bên dưới các khái niệm quen thuộc như phần mềm, phần cứng máy tính. Mô hình tư duy này sẽ cung cấp các khung kiến thức để giải quyết các bài toán, vấn đề nảy sinh. Đi cùng mô hình tư duy này là tập hợp các kiến thức lõi, cơ bản về lý thuyết cũng như thực hành, kỹ năng, năng lực phân tích, mô phỏng và giải quyết vấn đề.

Khoa học máy tính đi sâu vào tìm hiểu cách làm việc và vận hành của máy tính và các hệ thống máy tính, tìm hiểu các máy tính, chương trình được thiết kế và lập trình như thế nào. Học sinh sẽ được tiếp cận với các hệ thống tính toán theo mọi khía cạnh, có thể cần hoặc không cần có máy tính. **"Tư duy máy tính"** sẽ có ảnh hưởng đến cả các lĩnh vực khoa học khác như vật lý, sinh học, hóa học, ngôn ngữ, tâm lý học, kinh tế và thống kê. Cũng chính tư duy đó sẽ giúp học sinh có thể giải các bài toán, giải quyết vấn đề được đặt ra, thiết kế hệ thống, thiết kế sản phẩm, hiểu được sức mạnh cũng như giới hạn, hạn chế của con người và máy móc. Chính tư duy đó sẽ là yêu cầu năng lực chính mà mỗi học sinh cần hiểu và nắm bắt được dù chỉ một phần của nó. Nếu có các kỹ năng, tư duy, suy luận như "máy tính" thì học sinh sẽ hiểu tốt hơn và có nhiều cơ hội hơn trong việc tiếp thu các công nghệ "dựa trên máy tính" (computer-based technology) và sẽ được chuẩn bị tốt hơn khi trở thành công dân tương lai trong xã hội hiện đại.

Khoa học máy tính là môn học thực hành, trong đó rất khuyến khích sự dũng cảm và sáng tạo. Học sinh được học các nguyên tắc, lý thuyết hàn lâm của môn học và khuyến khích ứng dụng sáng tạo để mô phỏng được thế giới thực xung quanh các em. Sự kết hợp hài hòa giữa lý thuyết, thực hành và sáng tạo sẽ làm cho môn học này trở nên vô cùng hấp dẫn, giúp các học sinh có thể tạo ra được các sản phẩm vừa có ích ("Nó chạy rồi!") vừa trí tuệ ("Nó quá đẹp!").

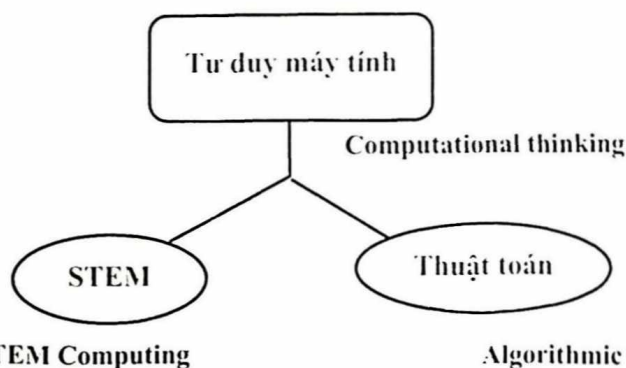
Tư duy máy tính ở đây cần được hiểu là nói về khả năng con người có thể làm, chứ không phải máy tính có thể làm. Ví dụ khi nhắc đến tư duy máy tính, người ta thường nhắc đến các khả năng suy nghĩ và làm việc logic, có tính (tối ưu) thuật toán, có thể lặp lại và có thể trừu tượng hóa. Nói một cách đơn giản hơn, tư duy máy tính là khả năng của con người, sử dụng các tính năng ưu việt của máy tính để giải quyết các vấn đề thực tế của con người. Bảng sau cho chúng ta thấy rõ hơn sự khác biệt giữa 2 mạch kiến thức ICT và CS.

	ICT - CNTT và ứng dụng	CS - Khoa học máy tính
Hệ thống máy tính	Hệ thống máy tính được sử dụng như thế nào.	Hệ thống máy tính được hoạt động, làm việc như thế nào.
Đối tượng	Con người là trung tâm của môn học.	Máy tính là trung tâm của môn học.

	ICT - CNTT và ứng dụng	CS - Khoa học máy tính
Định hướng cốt lõi	Tập trung, quan tâm đến sự phát triển của hệ thống hướng tới nhu cầu người sử dụng.	Tập trung, quan tâm đến tư duy thuật toán, đến cách mà vấn đề có thể phân rã thành các bài toán, vấn đề nhỏ hơn để giải quyết.
Sản phẩm	Quan tâm đến việc sử dụng các phần mềm, hệ thống đã có để đáp ứng nhu cầu hiện tại.	Quan tâm đến việc thiết kế các hệ thống, phần mềm mới.
Sử dụng sản phẩm	Nhấn mạnh việc lựa chọn, đánh giá sử dụng phần mềm trong công việc.	Nhấn mạnh đến các nguyên lý và kỹ thuật của hệ thống, phần mềm. Lập trình luôn đóng vai trò trung tâm của các vấn đề quan tâm.
Tư duy hệ thống	Hệ thống ứng dụng CNTT phải hỗ trợ hoạt động của con người hay tự động hóa hoạt động của con người.	Các ứng dụng thực tế cần được xây dựng thông qua các "tư duy của máy tính". Thông qua "tư duy" này chúng ta sẽ hiểu được thế giới tự nhiên như bản chất nó có, nhưng theo cách hiểu của chúng ta, thông qua máy tính.
Định hướng chung	Định hướng ứng dụng, nghề nghiệp.	Định hướng chuyên nghiệp, hàn lâm.

Như vậy trong phân môn **Khoa học máy tính** (CS), học sinh cần được trang bị các kiến thức, công cụ lập trình để giải quyết các vấn đề của mình. Bên cạnh tư duy thuật toán là định hướng kiến thức lõi của Khoa học máy tính như chúng ta vẫn hiểu, còn một hướng kiến thức phát triển quan trọng nữa, đó là định hướng làm ra các sản phẩm cụ thể phục vụ các mục đích khác nhau của cuộc sống hàng ngày. Định hướng này chính là STEM COMPUTING, có thể hiểu chính là định hướng giáo dục STEM của môn Tin học.

Mô hình của Tư duy máy tính của môn Tin học mới được mô tả trong sơ đồ đơn giản sau:



Chúng ta cùng xem lại trong chương trình môn Tin học mới 2018, mạch kiến thức Khoa học máy tính được phân bổ qua các cấp học như thế nào.

Cấp học	Phân bổ kiến thức	Ngôn ngữ lập trình
Tiểu học	- Làm quen với ngôn ngữ lập trình trực quan để làm ra các sản phẩm STEM đơn giản, các trò chơi đơn giản.	Có thể chọn Scratch hoặc một ngôn ngữ lập trình trực quan bất kỳ.
THCS	- Bắt đầu làm quen với khái niệm thuật toán. - Tìm hiểu sâu hơn ngôn ngữ lập trình trực quan để tạo ra các sản phẩm hoặc phần mềm hoàn chỉnh hơn.	Có thể chọn ngôn ngữ lập trình Scratch hoặc một ngôn ngữ lập trình tương tự bất kỳ.
THPT (chuyên ngành CS)	- Làm quen với một ngôn ngữ lập trình bậc cao. - Các chuyên đề lắp ráp và lập trình Robot. - Các chuyên đề sâu hơn về thuật toán và cấu trúc dữ liệu đơn giản.	Có thể chọn một ngôn ngữ lập trình bậc cao bất kỳ như Pascal, C, Java, Python, ...

Chúng ta thấy gì từ bảng trên?

- Định hướng CS - Khoa học máy tính trong môn Tin học mới đã được đưa xuống cho học sinh Tiểu học, bắt đầu từ lớp 3 và mạch kiến thức này sẽ được học liên tục cho đến lớp 12.

- Từ cấp Tiểu học, học sinh sẽ được làm quen với môi trường lập trình trực quan, đơn giản, bước đầu làm quen, dần được trải nghiệm và biết cách tự thiết lập cho mình một chương trình hay trò chơi đơn giản. Như vậy ở cấp Tiểu học, định hướng STEM sẽ được chú trọng hơn.

- Lên cấp THCS, học sinh sẽ làm quen với khái niệm thuật toán, các cấu trúc lệnh lập trình hoàn chỉnh và hoàn thiện, nâng cao khả năng STEM của mình thông qua việc làm ra các sản phẩm có ích cho cuộc sống. Ở cấp THCS, cả 2 hướng STEM và Algorithm cùng được chú trọng ngang nhau.

- Lên cấp THPT, học sinh sẽ được học chính thức một ngôn ngữ lập trình bậc cao. Các chuyên đề về lắp ráp, lập trình Robot và thuật toán nâng cao sẽ được đưa vào chương trình như các lựa chọn bắt buộc.

Như vậy bắt đầu từ cấp THCS, cả 2 hướng STEM và Thuật toán đều được chú trọng trong mạch kiến thức CS của Tin học. Tới đây chúng ta sẽ thấy vai trò của Python là rất thích hợp. Python là ngôn ngữ lập trình bậc cao với cách viết lệnh đơn giản theo từng dòng sẽ rất phù hợp để học và dạy thuật toán. Python cùng với các module đồ họa con Rùa và Tkinter có cách tiếp cận khá đơn giản với các ứng dụng đồ họa nên dễ dàng hơn hẳn các ngôn ngữ khác để thiết lập các ứng dụng kiểu STEM. Chính vì điều này Python sẽ là ngôn ngữ thích hợp được lựa chọn cho học sinh làm quen bắt đầu từ cấp THCS và THPT trong chương trình môn Tin học mới.

Cuốn sách này ra đời nhằm đáp ứng các nhu cầu đã nêu trên. Đây là cuốn sách đầu tiên, Python cơ bản, dành cho người mới bắt đầu học ngôn ngữ lập trình này.

Sách bao gồm 16 chương, cùng với trên 350 bài tập từ đơn giản đến nâng cao, phù hợp cho mọi đối tượng từ cấp THCS, THPT hoặc sinh viên đại học. Sách cũng có thể dùng cho giáo viên dạy Tin học các trường phổ thông và đại học.

Nội dung dự kiến các chủ đề - chương - bài học Lập trình Python cơ bản sẽ như sau. Mỗi chương có thể coi như một bài học hoặc có thể là một dãy các bài tự học.

1. Bắt đầu với Python.
2. Làm quen môi trường lập trình Python.
3. Input và chuyển đổi dữ liệu.
4. Hàm số.
5. Đối tượng trong Python.
6. Kiểu dữ liệu List. Mảng một chiều.
7. List của List. Mảng nhiều chiều.
8. Khái niệm Module.
9. Xâu ký tự.
10. Đọc và ghi tệp.
11. Câu lệnh điều kiện.
12. Đệ quy.
13. Kiểu dữ liệu Từ điển.
14. Kiểu dữ liệu Tập hợp.
15. Đồ họa con Rùa.
16. Bắt lỗi và kiểm soát lỗi trong Python.

Mỗi chương, bài học sẽ bắt đầu bằng mô tả mục đích của chương, tiếp theo là dãy các hoạt động kiến thức cần học và dạy. Sách có thể dùng cho việc tự học hoặc giáo viên giảng dạy trên lớp. Sau mỗi chương là phần câu hỏi, bài tập chi tiết.

Sách bao gồm các phần Lời nói đầu; phần các chương nội dung chính; cuối sách là Index và Tài liệu tham khảo.

Với học sinh THCS và THPT, nội dung sách có thể được dạy trong 2 học kỳ với thời lượng khoảng 120 tiết, hoặc tương đương một học phần 4 - 6 tín chỉ với sinh viên các trường đại học kỹ thuật.

Bùi Việt Hà

MỤC LỤC

Lời nói đầu	3
MỤC LỤC	10
Chương 1. Bắt đầu với Python	15
Nội dung chính	15
1. Lịch sử ngôn ngữ lập trình Python	15
2. Cài đặt Python lỗi	15
3. IDE - giao diện soạn thảo lập trình Python	16
4. In dòng chữ "Hello World" - lệnh đầu tiên	16
5. Hello World.py - chương trình đầu tiên	17
6. Phần mềm soạn thảo lập trình Python	18
7. Trợ giúp, comment và dấu ngăn cách lệnh	22
Câu hỏi - bài tập - thực hành	24
Chương 2. Làm quen với môi trường tương tác Python	26
Nội dung chính	26
1. Kiểu dữ liệu trong Python	26
2. Biểu thức trong Python	27
3. Biến nhớ và lệnh gán	30
4. Từ khóa trong Python	32
5. Các phép tính nâng cao khác	32
6. Namespace	34
Câu hỏi - bài tập	36
Chương 3. Input và chuyển đổi dữ liệu	39
Nội dung chính	39
1. Input dữ liệu	39
2. Chuyển đổi trực tiếp dữ liệu	40
3. Hàm eval() tính toán giá trị từ xâu biểu thức	42
4. Kiểu bool và lệnh if	43
5. Lệnh lặp for	46
6. Lệnh lặp while	47
Câu hỏi - bài tập	48
Chương 4. Hàm số	51
Nội dung chính	51

1. Khái niệm hàm số trong Python	51
2. Định nghĩa hàm số	51
3. Kiểu truyền giá trị của tham số trong hàm số	53
4. Phạm vi áp dụng hàm số	54
5. Phạm vi của biến nhớ và tham số trong hàm số	54
6. Tham số thực sự và tham số lựa chọn	56
7. Hàm định nghĩa trực tiếp	58
8. Một vài ví dụ hàm số	58
Câu hỏi - bài tập.....	59
Chương 5. Đối tượng.....	63
Nội dung chính	63
1. Tổng quan về lập trình hướng đối tượng (OOP)	63
2. Khởi tạo Lớp và Đối tượng	64
3. Lệnh dir() và __init__()	66
4. Các toán tử chồng (Overloaded Operator).....	69
5. Lớp con và tính kế thừa.....	71
Câu hỏi - bài tập.....	73
Chương 6. Kiểu dữ liệu List. Mảng một chiều	75
Nội dung chính	75
1. Khởi tạo List.....	75
2. Các phép toán và phương thức của List	76
3. Lệnh for trên dữ liệu List	79
4. Hàm range và chỉ số của dãy List.....	80
5. Kiểu dữ liệu Tuple.....	82
6. Một vài bài toán cụ thể	83
Câu hỏi - bài tập.....	85
Chương 7. List của List. Mảng nhiều chiều.....	89
Nội dung chính	89
1. Các vòng lặp lồng nhau.....	89
2. List của List	90
3. Mô tả ma trận trong Python.....	91
4. List of List - dữ liệu có cấu trúc.....	92
5. Chỉ số âm và vùng chỉ số (Slicing)	96
6. Liên kết tham biến của List	98
7. Chức năng nâng cao: list comprehension	102

8. Đối tượng dữ liệu tuần tự (iterable)	102
Câu hỏi - bài tập.....	105
Chương 8. Khái niệm Module.....	109
Nội dung chính	109
1. Lệnh import	110
2. Module math.....	111
3. Số phức và module cmath.....	111
4. Tự thiết lập module	113
5. Sinh số ngẫu nhiên trong Python	114
6. Sinh giá trị ngẫu nhiên từ một dãy	115
Câu hỏi - bài tập.....	116
Chương 9. Xâu ký tự.....	119
Nội dung chính	119
1. Kiểu dữ liệu String	119
2. Bảng mã chuẩn ASCII và Unicode	120
3. Dãy thoát Escape	124
4. Các phương thức tự động điều chỉnh và căn chỉnh xâu ký tự.....	126
5. Lệnh tạo khuôn format()	132
Câu hỏi - bài tập.....	143
Chương 10. Đọc và ghi tệp.....	148
Nội dung chính	148
1. Các lệnh đọc thông tin từ tệp.....	150
2. Các lệnh ghi thông tin ra tệp	153
3. Một số ứng dụng đọc và ghi tệp.....	157
4. Tổng kết các phương thức đọc và ghi tệp.....	158
Câu hỏi - bài tập.....	159
Chương 11. Câu lệnh điều kiện.....	163
Nội dung chính	163
1. Kiểu dữ liệu bool.....	163
2. Biểu thức logic và các toán tử so sánh	166
3. Lệnh if	168
4. Lệnh lặp while.....	170
5. break và continue	171
6. Toán tử in	173
7. Nguyên tắc chuyển mạch nhanh của các phép toán logic	173

Câu hỏi - bài tập.....	176
Chương 12. Dệ quy.....	180
Nội dung chính	180
1. Khái niệm hàm đệ quy.....	180
2. Thiết lập đệ quy đúng.....	182
3. Một vài ứng dụng đệ quy	185
Câu hỏi - bài tập.....	188
Chương 13. Kiểu dữ liệu từ điển.....	194
Nội dung chính	194
1. Các lệnh keys() và values().....	195
2. Lệnh items().....	197
3. Lệnh update()	197
4. Lệnh get().....	197
5. Xóa khóa và dữ liệu của từ điển	198
6. Một số ứng dụng đơn giản của dữ liệu từ điển	199
Câu hỏi - bài tập.....	200
Chương 14. Kiểu dữ liệu tập hợp.....	205
Nội dung chính	205
1. Khởi tạo tập hợp.....	205
2. Thêm bớt phần tử của tập hợp	208
3. Các phép toán trên tập hợp	209
4. Tổng kết các kiểu dữ liệu trên Python đã học.....	211
Câu hỏi - bài tập.....	212
Chương 15. Đồ họa con rùa.....	216
Nội dung chính	216
1. Thiết lập môi trường đồ họa Turtle.....	216
2. Hướng của rùa, chế độ vẽ và tọa độ màn hình	217
3. Các lệnh điều khiển rùa trên màn hình	218
4. Thiết lập kích thước sân khấu và hệ tọa độ	219
5. Thiết lập bút vẽ và tô màu các hình khối.....	222
6. Thiết lập chế độ tương tác đơn giản trên màn hình	225
7. Tìm hiểu sâu hơn về điều khiển rùa	231
Câu hỏi - bài tập.....	235
Chương 16. Bắt lỗi và kiểm soát lỗi chương trình.....	238
Nội dung chính	238

1. Lỗi syntax và lỗi logic nội tại	238
2. Lỗi Exception (ngoại lệ)	239
3. Bắt lỗi Exception với lệnh try, mức đơn giản	239
4. Bắt lỗi Exception, mức đầy đủ.....	241
5. Bắt lỗi chi tiết.....	243
6. Một vài lỗi Exception có sẵn trong Python.....	245
7. Tự tạo lỗi (raise Exception).....	245
8. Mô hình đối tượng của lỗi Exception	247
Câu hỏi - bài tập.....	249
INDEX	251
Tài liệu tham khảo.....	253

Chương 1. Bắt đầu với Python

Mục tiêu:

- Làm quen với ngôn ngữ lập trình Python.
- Làm quen với môi trường lập trình tương tác của Python.



Nội dung chính

1. Lịch sử ngôn ngữ lập trình Python

Python thuộc loại ngôn ngữ lập trình bậc cao. Python được công bố lần đầu tiên vào tháng 12 năm 1989. Tác giả của Python là **Guido van Rossum**, người Hà Lan, tại Centrum Wiskunde & Informatica (CWI).

Logo của Python.



Bản Python chính thức 1.0 được phát hành năm 1991.

Python 2.0 ra đời năm 2000.

Python 3.0 ra đời năm 2008 và liên tục phát triển cho đến hôm nay.

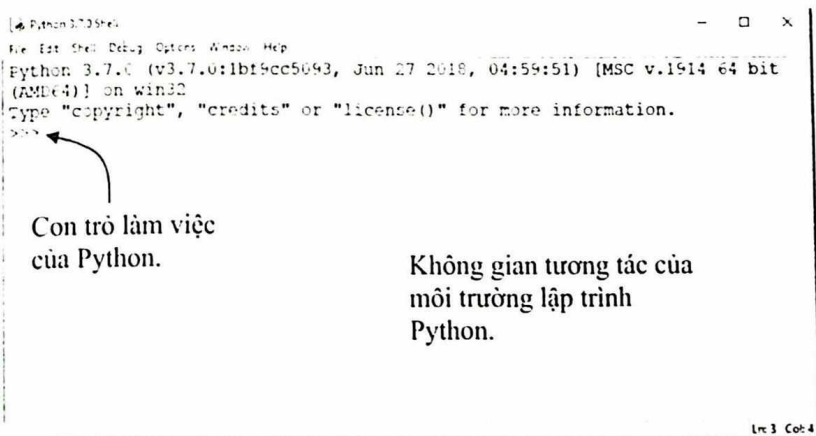
Bản thân Python là phần mềm mã nguồn mở với giấy phép GPL. Trang Web chính thức của ngôn ngữ này là *Python.org*.

Hiện nay Python là một trong những ngôn ngữ lập trình bậc cao phổ biến nhất trên thế giới, phù hợp cho mọi lứa tuổi.

2. Cài đặt Python lỗi

Khi cài đặt Python lỗi chúng ta sẽ có ngay bản **Python Shell** (giao diện tương tác) đi kèm một chương trình soạn thảo Editor đơn giản.

Giao diện của Python Shell như hình sau. Các bạn chú ý đến dấu `>>>` là con trỏ làm việc đặc trưng riêng của Python.



Chúng ta chú ý đến khu vực làm việc của Python Shell và vị trí con trỏ làm việc của Python >>>. Các lệnh có thể gõ trực tiếp sau con trỏ này. Đây là chế độ làm việc tương tác của Python.

Ví dụ một số lệnh trực tiếp tính toán như sau.

```
>>> 12+124
136
>>> 78*309
24102
>>> 25**2
625
>>>
```

****** là phép toán lũy thừa. $25^{**}2 = 25^2$

Nếu lệnh gõ đúng kết quả sẽ hiển thị ngay bên dưới. Ngược lại nếu lệnh được nhập sai thì thông báo lỗi sẽ xuất hiện.

Muốn thoát khỏi Python Shell gõ tổ hợp phím Ctrl-Q.

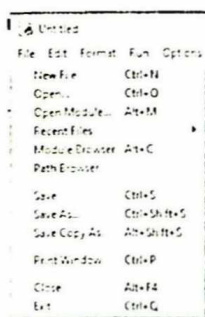
3. IDE - giao diện soạn thảo lập trình Python

Ngoài môi trường tương tác Python Shell, Python còn cho phép soạn thảo chương trình trong các tệp dạng *.py.

Từ giao diện Python Shell dễ dàng tạo và thiết lập các tệp chương trình này thông qua một Editor đơn giản có sẵn của Python Shell. Giao diện của Python Editor như hình sau.



Các lệnh làm việc với tệp chương trình *.py.



Như vậy để lập trình trên Python có thể dùng trình Editor của Python lỗi, hoặc có thể dùng các chương trình soạn thảo khác (xem phần sau).

Các lệnh Python mặc định được viết trên từng dòng của tệp. Một trong những đặc điểm khác biệt nhất của Python là cách viết lệnh theo nguyên tắc "thụt vào". Chúng ta sẽ tìm hiểu các nguyên tắc này trong quá trình học các lệnh tiếp theo của Python.

4. In dòng chữ "Hello World" - lệnh đầu tiên

Chúng ta sẽ bắt đầu làm quen với lệnh đầu tiên của Python, đó là lệnh **print**. Chú ý Python phân biệt chữ hoa, chữ thường, do vậy **print** khác với **Print**.

Ví dụ lệnh sau sẽ in dòng chữ "Hello World !!" trên màn hình.

```
>>> print("Hello World !!")
Hello World !!
```

Xâu ký tự ghi trong tham số của lệnh **print** có thể viết dưới dạng "Hello World !!" hoặc 'Hello World !!'.

Nếu viết sai lệnh, ví dụ nếu gõ lệnh "Print", lỗi sẽ xảy ra và thông báo ngay, như sau.

```
>>> Print("Hello World !!")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    Print("Hello World !!")
NameError: name 'Print' is not defined
>>>
```

Trong môi trường tương tác **Python Shell**, lệnh được viết ra ngay sau vị trí con trỏ **>>>**.

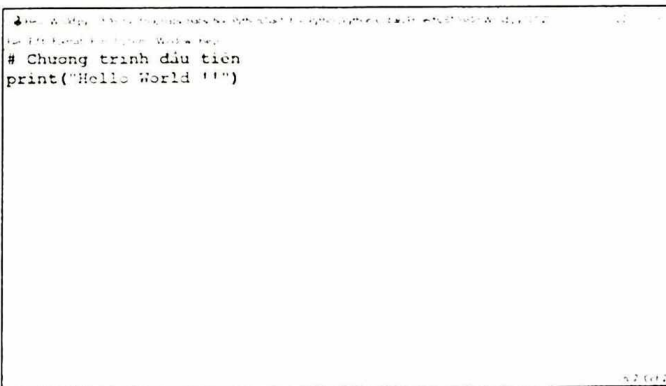
5. Hello World.py - chương trình đầu tiên

Chúng ta sẽ viết một chương trình hoàn chỉnh đầu tiên trên Python. Các tệp chương trình của Python đều có phần mở rộng *.py và được soạn thảo bởi các phần mềm IDE bất kỳ, miễn sao hỗ trợ Python.

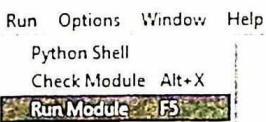
Hello World.py

```
1. # Chương trình đầu tiên
2. print("Hello World !!")
```

Ví dụ tệp này được soạn thảo bằng Editor của Python, giao diện soạn thảo như hình sau.



Để chạy chương trình này, chúng ta thực hiện lệnh **Run** → **Run Module** hoặc bấm **F5**.



Kết quả chạy của chương trình được thể hiện trên giao diện tương tác của Python Shell là đầu ra chuẩn như sau.

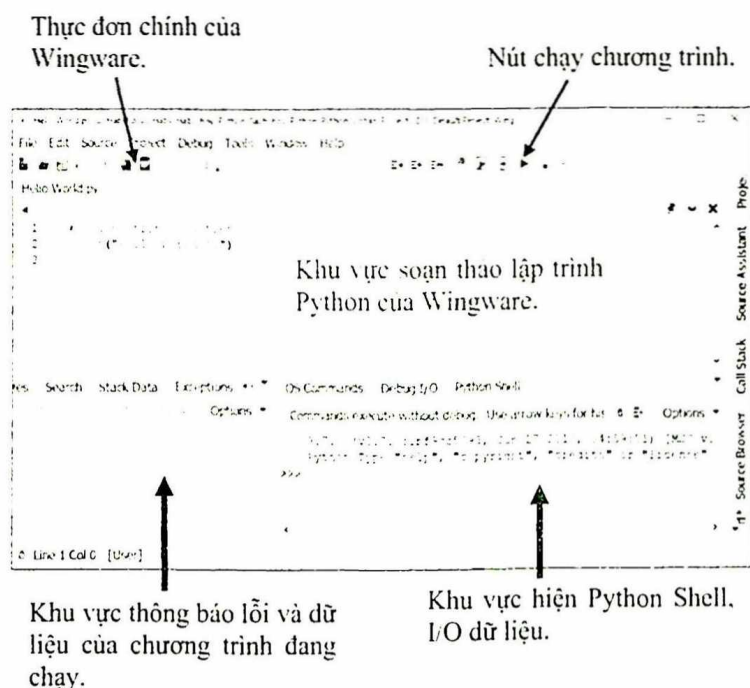
```
>>>
RESTART:           D:\Habv\docs\Habv\Habv_hoc_Python\Sach-hoc-
Python\Python-Coban\Projects\B1\Hello World.py
Hello World !!
>>>
```

6. Phần mềm soạn thảo lập trình Python

6.1. Wingware Personal

Đối với những ai làm việc chuyên nghiệp hơn trên Python, một chương trình soạn thảo lập trình chuyên nghiệp hơn trên Python sẽ cần đến. Chúng ta sẽ cùng làm quen với một phần mềm như vậy, đó là chương trình soạn thảo Wingware. Phiên bản miễn phí khá chuyên nghiệp là **Wingware Personal**. Phần mềm này khá nhỏ, đơn giản, dễ sử dụng, tích hợp hoàn toàn với Python lỗi.

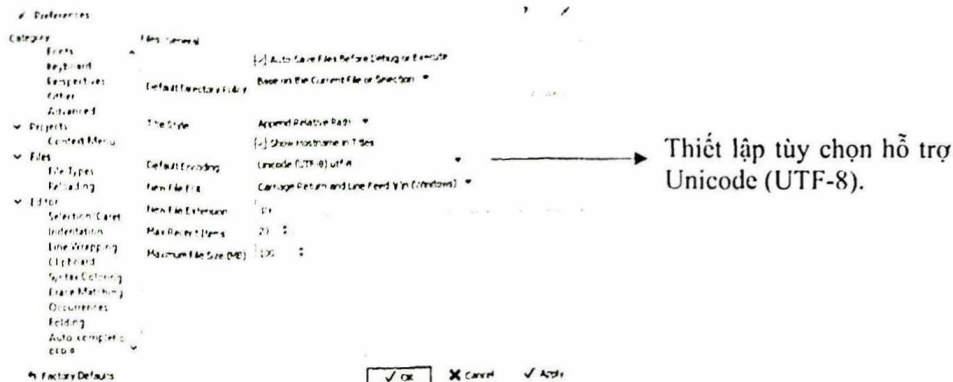
Sau đây là giao diện của Wingware Personal.



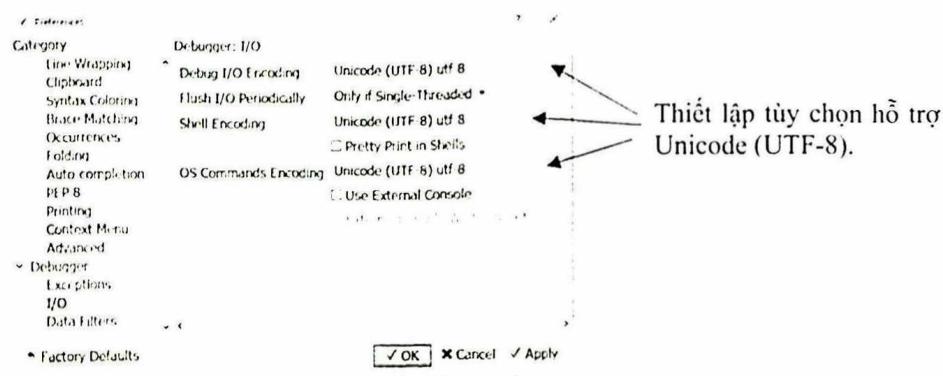
Chú ý:

Mặc định, các tệp chương trình của Python (tệp *.py) là dạng ASCII, tuy nhiên cũng có thể thiết lập tùy biến cho các tệp chương trình hỗ trợ Unicode.

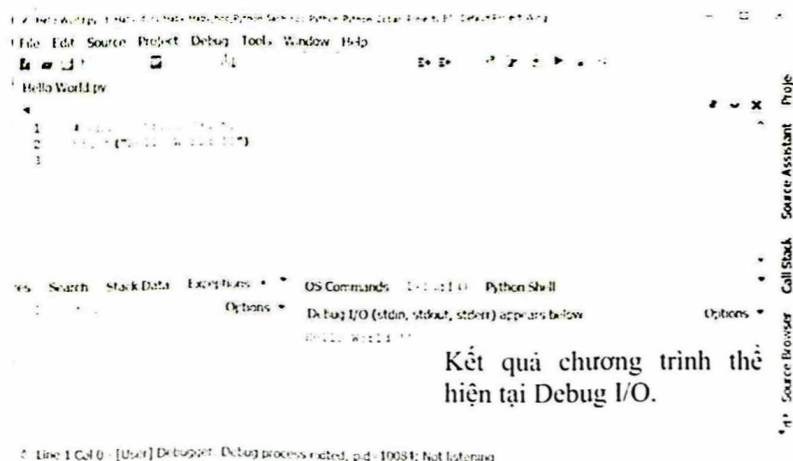
Các tùy biến cho thể hiện phông chữ Unicode trên trình soạn thảo, dữ liệu vào ra được thiết lập từ lệnh Edit → Preferences → Files.



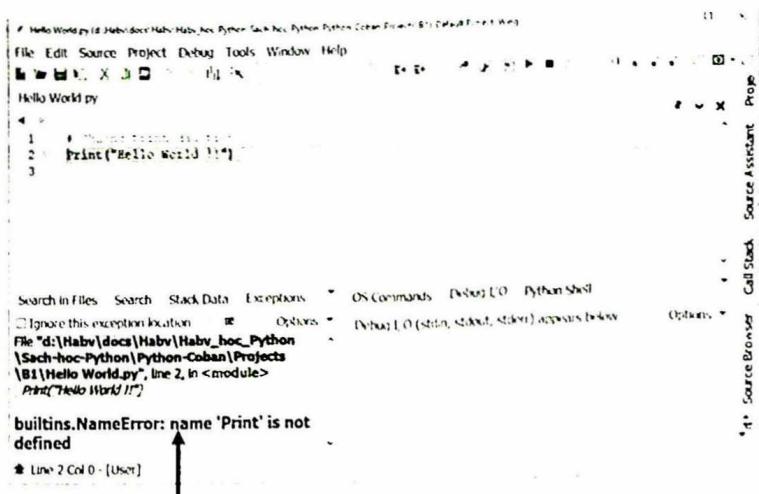
Thiết lập hỗ trợ Unicode cho các cửa sổ I/O như sau:



Ví dụ chương trình đầu tiên **Hello World.py** được chạy trên Wingware như sau.



Nếu có lỗi thì cửa sổ sẽ thể hiện tất cả các lỗi của chương trình như sau.



Vị trí dòng lệnh có lỗi được đánh dấu ngay trong cửa sổ soạn thảo.

Khu vực thể hiện lỗi và kiểu lỗi chương trình.

6.2. PyCharm

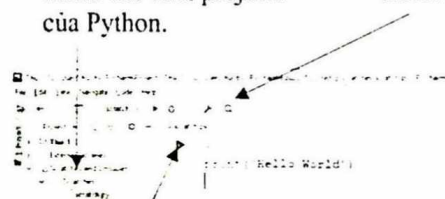
PyCharm là công cụ soạn thảo lập trình Python nổi tiếng nhất hiện nay. PyCharm là sản phẩm của hãng Jet Brains, công ty số một thế giới hiện nay về các công cụ IDE lập trình hiện đại. Phiên bản PyCharm Edu được phát hành miễn phí.



PyCharm có giao diện rất đơn giản, dễ sử dụng.

Khu vực xem và điều khiển các file, projects của Python.

Hệ thống thực đơn, thanh công cụ chính.



Nút chạy chương trình.

Cửa sổ soạn thảo lập trình.



6.3. PyScripter



PyScripter là phần mềm IDE cho Python rất đơn giản, mã nguồn mở và hoàn toàn miễn phí. Phần mềm do tác giả Kiriakos Vlahos viết trên Delphi. Chương trình này rất nhỏ gọn, phù hợp cho môi trường các nhà trường và cũng có đầy đủ các chức năng của một môi trường lập trình, gỡ rối Python đầy đủ. Giao diện chính của PyScripter như sau.

Khu vực này điều khiển các projects, thư mục và code.

Hệ thống thực đơn, thanh công cụ chính.

Nút chạy chương trình.

Khu vực soạn thảo lập trình chính.

Cửa sổ Python Shell.

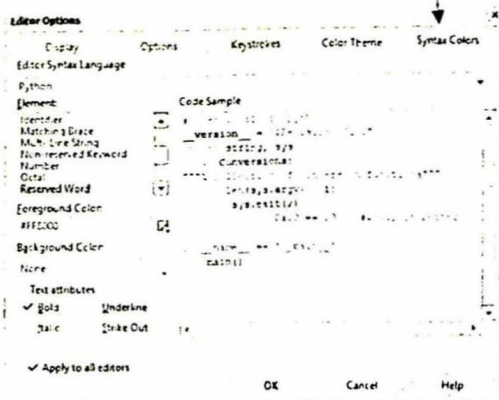
Lệnh View → Options → Editor Options sẽ cho phép thiết lập các chế độ hiển thị khác nhau của cửa sổ soạn thảo.



Thiết lập font và cỡ chữ cho cửa sổ soạn thảo.



Thiết lập màu sắc cho cách thể hiện các thành phần khác nhau của ngôn ngữ lập trình.



7. Trợ giúp, comment và dấu ngăn cách lệnh

7.1. Lệnh trợ giúp `help()`

Muốn xem trợ giúp cú pháp hay ý nghĩa của một lệnh bất kỳ của Python chúng ta dùng lệnh `help()`. Có 2 cách thực hiện như sau.

Cách 1: Thực hiện trực tiếp lệnh `help(<tên lệnh cần xem trợ giúp>)`.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout,
          flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current
    sys.stdout.
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Cách 2: Gõ lệnh `help()`. Khi đó con trỏ trợ giúp `help>` xuất hiện. Chúng ta có thể gõ tên lệnh muốn xem trợ giúp ngay sau con trỏ này, ví dụ.

```
>>> help()
.
.
help>

help> print
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout,
          flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current
    sys.stdout.
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

help>
```

Muốn thoát chế độ con trỏ trợ giúp `help>` chỉ cần bấm Enter sau con trỏ này.

Như vậy cú pháp đầy đủ của lệnh `print` là:

```
print(value1, value2, ..., valueN, sep = <ký tự, dấu ngăn cách>,
end = <xâu kết thúc lệnh>, file = <đối tượng tệp ra của lệnh>,
flush = <true/false>)
```

Trong đó:

value1, value2, ..., valueN là các giá trị cần in ra, các giá trị cần in có thể là số, xâu ký tự, biểu thức hay biến nhớ.

sep = ký tự, dấu ngăn cách giữa các giá trị value1, value2, ..., valueN ở trên. Mặc định giá trị này là dấu cách, tức là sep = ' '.

end = ký tự, dấu kết thúc của lệnh print. Mặc định giá trị này là xâu điều khiển xuống dòng (kết thúc 1 dòng), tức là end = '\n'.

Các tham số **file** và **flush** chúng ta sẽ tìm hiểu sau.

7.2. Ngăn cách lệnh

Python là chương trình viết mỗi lệnh trên một dòng.

Tuy nhiên, về nguyên tắc chúng ta cũng có thể gõ nhiều lệnh trong một dòng lệnh, khi đó các lệnh cần kết thúc bởi dấu ngăn cách ";" tương tự như các ngôn ngữ lập trình khác.

Ví dụ:

```
>>> print("Hà");print("Nội")
Hà
Nội
>>>
```

Thông thường chương trình Python sẽ viết mỗi lệnh trên một dòng, khi đó không cần các ký tự ngăn cách lệnh nữa.

7.3. Ghi chú trong chương trình

Ghi chú (comment) lệnh trong Python được bắt đầu bằng ký tự # và có tác dụng từ vị trí này đến cuối dòng hiện thời.

Ví dụ một cách ghi chú lệnh trong Python.

```
>>> x = 2          # gán 2 cho biến x
>>> print(x)
2
>>>
```

Còn đây là một ghi chú sai.

```
>>> /* ghi chú */
SyntaxError: invalid syntax
>>> x = 2 /* ghi chú */
SyntaxError: invalid syntax
>>>
```

7.4. Viết lệnh trên nhiều dòng

Chúng ta có thể viết một lệnh dài trên nhiều dòng. Muốn xuống dòng cần gõ ký hiệu "\" sau đó có thể nhấn Enter để xuống dòng viết tiếp dòng lệnh đó.

Ví dụ:

```
>>> a = "Hà Nội là thủ đô \
của Việt Nam"
>>> b = 1 + 2/3 + 2.0000034 + \
        23.6 - 12/3
>>> b
23.26667006666667
```



Câu hỏi - bài tập - thực hành

1. Tải phần mềm Wing từ site <https://wingware.com/> và cài đặt chương trình để sẵn sàng cho việc lập trình Python trong môi trường này.
2. Thiết lập môi trường làm việc Wingware Personal của bạn hỗ trợ hoàn toàn tiếng Việt (cả trong Editor và Python Shell).
3. Lệnh `print("Hà Nội", "Việt Nam")` sẽ in ra kết quả như thế nào?
 - A. "Hà Nội" "Việt Nam"
 - B. Hà Nội Việt Nam
 - C. Hà NộiViệt Nam
 - D. Hà Nội
Việt Nam
4. Nhóm lệnh `print("Hà Nội"); print("Việt Nam")` sẽ in ra kết quả như thế nào?
 - A. "Hà Nội" "Việt Nam"
 - B. Hà Nội Việt Nam
 - C. Hà NộiViệt Nam
 - D. Hà Nội
Việt Nam
5. Lệnh `print('Xin chào bạn')` sẽ in ra kết quả như thế nào?
6. Tải phần mềm PyCharm từ link: <https://www.jetbrains.com/pycharm/>, cài đặt và chạy thử, so sánh với Wingware Personal.
7. Tải phần mềm PyScripter từ link sau:
<https://sourceforge.net/projects/pyscripter/>
Cài đặt, chạy thử và so sánh với 2 phần mềm soạn thảo Python đã biết là Wingware và PyCharm.
8. Các lệnh gán sau đây là đúng hay sai trong Python:
 - A. `x = print(x)`
 - B. `x = "1234" + 567`
 - C. `x = 7, y = 10`
 - D. `x = y = z = 10`

E. `print = print(123)`

F. `abc = print(0)`

9. Biên soạn chương trình đơn giản sau lần lượt trên các trình soạn thảo Wingware, PyCharm và PyScripter.

Hello.py

```
# Chương trình đầu tiên
x = "Hello World!"
print(x)
```

Chạy chương trình trên.

10. Theo bạn có thể copy các lệnh từ tệp *.py vào cửa sổ Python Shell được hay không?

11. Trong các lệnh sau, lệnh nào là lệnh Python đúng?

A. `print("Hello World")`

B. `Print("Hello World")`

C. `print(Hello World)`

D. `print("Hello World")`

E. `Print('Hello World')`

12. Để viết chú thích cho lệnh trong Python người ta dùng ký tự nào dưới đây:

A. Ký tự `//`

B. Ký tự `/* */`

C. Ký tự `*`

D. Ký tự `#`

13. Có thể lập trình trực tiếp trong môi trường tương tác Python Shell được không?

14. Nếu copy nội dung một tệp chương trình *.py vào cửa sổ Python Shell thì có chạy được không?

15. Có thể gõ dấu cách phía trước một lệnh Python được không?

16. Viết chương trình in ra màn hình bài thơ sau:

Mình về mình có nhớ ta?

Mười lăm năm ấy thiết tha mặn nồng.

Mình về mình có nhớ không?

Nhìn cây nhớ núi, nhìn sông nhớ nguồn.

17. Viết chương trình trên Python tính tổng $1 + 2 + 3 + \dots + 10$.

18. Viết chương trình trên Python tính biểu thức $1^1 + 2^2 + 3^3 + 4^4 + 5^5$.

19. Lệnh `print("Hà Nội", "thủ đô", "Việt Nam", sep = "\n", end = " ")` sẽ in ra kết quả như thế nào?

20. Các lệnh đã nhập trong Python Shell có thể sao chép sang cửa sổ soạn thảo lập trình được không?

Chương 2. Làm quen với môi trường tương tác Python

Mục tiêu:

Làm quen với các lệnh, khái niệm, cơ sở chính của ngôn ngữ lập trình Python.

- Kiểu dữ liệu.
- Lệnh.
- Từ khóa.
- Biến nhớ.
- Lệnh gán.
- Các phép tính toán học.



Nội dung chính

1. Kiểu dữ liệu trong Python

a. Lệnh `type()` và kiểu dữ liệu chính trong Python

Lệnh `type(<data>)` sẽ trả lại kiểu dữ liệu hiện thời của `<data>`.

Quan sát và thực hiện các lệnh sau.

```
>>> type(2.4)
<class 'float'>
>>> type("Hanoi")
<class 'str'>
>>> type(3>2)
<class 'bool'>
>>> type(123)
<class 'int'>
>>>
```

Các kiểu dữ liệu thể hiện sau lệnh `type()`.
float, str, bool, int

Các kiểu dữ liệu cơ bản trong Python bao gồm:

- **int**: số nguyên.
- **float**: số thực với dấu phẩy động.
- **bool**: logic với 2 giá trị **True** và **False**.
- **str**: chuỗi ký tự.

b. Biến nhớ không cần khai báo và xác định kiểu dữ liệu

Một đặc điểm đặc biệt của Python là các biến nhớ không cần khai báo và không cần xác định kiểu dữ liệu trước.

Cùng xem đoạn chương trình tương tác sau.

```
>>> x
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    x
NameError: name 'x' is not defined
```

```
>>> x = 1
>>> x
1
>>>
```

Ở dòng đầu tiên khi gõ x, Python sẽ không nhận ra vì x chưa được gán bất cứ giá trị nào. Ở lần tiếp theo khi đã gán x = 1 thì biến nhớ x đã được xác định.

Trong Python, các biến nhớ chỉ xác định kiểu dữ liệu tại thời điểm gán giá trị, do đó biến nhớ không cần khai báo kiểu dữ liệu trước. Một biến nhớ có thể nhận nhiều kiểu giá trị khác nhau mà không cần khai báo lại.

Xem đoạn chương trình tương tác sau, biến nhớ x có thể nhận các giá trị với kiểu dữ liệu lần lượt là float, int, bool và str.

```
>>> x = 1.23
>>> type(x)
<class 'float'>
>>> x = 2
>>> type(x)
<class 'int'>
>>> x = x < 1
>>> type(x)
<class 'bool'>
>>> x = "123"+"456"
>>> type(x)
<class 'str'>
>>>
```

2. Biểu thức trong Python

a. Biểu thức số học

Biểu thức số học trong Python có thể được viết và tính dễ dàng như trong mọi ngôn ngữ lập trình bậc cao khác. Biểu thức có thể chứa giá trị số hoặc biến nhớ, hàm số có giá trị.

Xét các ví dụ sau.

```
>>> 15 + 2 # phép cộng
17
>>> 15 - 5 # phép trừ
10
>>> 2 * 5 # phép nhân
10
>>> 2 ** 5 # phép lũy thừa
32
>>> 2 ** 3 ** 2 # số mũ tính từ phải sang trái
512
>>> 5 / 2 # phép chia thực
2.5
>>> 5 // 2 # phép chia nguyên
2
>>> 5 % 2 # tính số dư của phép chia nguyên
1
>>>
```

Bảng các phép tính trong Python như sau:

Phép tính	Ý nghĩa	Phép tính	Ý nghĩa	Phép tính	Ý nghĩa
+, -	cộng, trừ	//	chia nguyên	**	lũy thừa
*	nhân	/	chia thực	%	lấy số dư

Chú ý:

- Các phép tính +, - được tính từ trái qua phải.

Ví dụ: $12 + 3 - 7 = (12 + 3) - 7 = 8$.

- Các phép tính *, /, // cũng được tính từ trái qua phải.

Ví dụ: $10 / 2 / 2 * 3 = ((10 / 2) / 2) * 3 = 7.5$.

- Trong biểu thức có cả +, - và *, /, //, % thì phép nhân, chia, % được ưu tiên hơn cộng, trừ.

Ví dụ: $2 + 2 / 2 = 3.0$.

- Riêng phép lũy thừa ** được tính từ phải qua trái.

Ví dụ:

```
>>> 2 ** 2 ** 3 # = 2 ** (2 ** 3)
256
>>> (2 ** 2) ** 3
64
```

- Phép tính lấy số dư % và chia nguyên // nếu số chia hoặc số bị chia là kiểu float thì kết quả sẽ chuyển đổi sang kiểu float. Ví dụ:

```
>>> 15.0 // 5
3.0
>>> 15 % 2.0
1.0
```

- Dãy các biểu thức có thể viết trên một dòng lệnh, cách nhau bởi dấu phẩy như sau:

```
>>> 3+4, 6-3, 7**2
(7, 3, 49)
```

Trong ví dụ trên nếu các biểu thức phân cách bởi dấu ";" thì Python sẽ hiểu là các lệnh khác nhau và kết quả sẽ như sau:

```
>>> 3+4; 6-3; 7**2
7
3
49
```

b. Phân biệt các phép chia / và //

- Trong Python, phép chia / luôn là chia thập phân. Kết quả phép tính luôn có kiểu float.

Ví dụ:

```
>>> 4/2
2.0
>>> 4.5/1.4
3.2142857142857144
```

- Phép chia // trong Python được gọi là **Floor Division**, hay có thể gọi chính xác **phép chia nguyên** (hay **chia trần**). Trong nhiều ngôn ngữ lập trình khác, toán tử chia nguyên này được ký hiệu **div**. Ý nghĩa của phép chia // như sau:

$a // b = \text{số nguyên lớn nhất không vượt quá } a / b.$

Nếu a hoặc b là float thì $a // b$ sẽ có kết quả float.

Nếu a và b là nguyên thì $a // b$ có kết quả nguyên.

Ví dụ:

```
>>> 5/2
2.5
>>> 5//2 # kết quả là số nguyên lớn nhất không vượt quá 2.5
2
>>> -3/2
-1.5
>>> -3//2 # kết quả là số nguyên lớn nhất không vượt quá -1.5
-2
>>> 3.0//2 # kết quả là kiểu số float.
1.0
>>>
```

c. Chú ý với phép tính lấy số dư %

Toán tử lấy số dư trong nhiều ngôn ngữ lập trình khác được ký hiệu là **mod**.

Toán tử % nếu cả 2 toán hạng là số nguyên dương thì phép tính là lấy số dư như bình thường.

```
>>> 3 % 2
1
>>> -3 % 2
1
>>> 4 % 6
4
>>> -4 % 6
2
>>> 3 % -2
-1
>>> 4 % -6
-2
>>>
```

Ta luôn có công thức sau đây mô tả quan hệ giữa 2 toán tử // và % trong phạm vi số nguyên:

$$a = b * (a // b) + (a \% b).$$

3. Biến nhớ và lệnh gán

a. Quy tắc đặt tên biến

Biến nhớ trong Python được quy định đặt tên cũng tương tự như các ngôn ngữ lập trình khác. Tên biến nhớ được quy định theo khái niệm định danh (**identifier**). Các định danh này dùng để đặt tên cho biến nhớ, kiểu dữ liệu, tên lớp, tên hàm, ...

Có thể tóm tắt quy tắc đặt tên biến nhớ trong Python như sau:

- Phân biệt chữ hoa và chữ thường.
- Chỉ có thể chứa ký tự là chữ, chữ số, dấu gạch dưới (**underscore**) và không cho phép chứa bất kỳ các ký tự khác.
- Tất cả các ký tự ngăn cách sau là không được phép có trong tên biến nhớ: dấu cách, các ký tự ngăn cách: `. : ! " ' # @ & ( )` và các ký tự thể hiện phép tính: `+ - * / ^ %`. Chỉ cho phép ký tự `_` (**underscore**) có trong định nghĩa biến nhớ.
- Tên biến nhớ phải bắt đầu bằng chữ cái hoặc dấu `_` (**underscore**), không được phép là chữ số.

Ví dụ các tên biến sau là hợp lệ: `ab_1`, `_bien_nho`, `xyz`, `x_y_z`.

b. Lệnh gán

Lệnh gán đơn giản nhất của Python có dạng:

`<biến nhớ> = <giá trị>`

Ví dụ:

```
>>> x = 1
>>> y = 10
>>> z = y + 5
>>> x, y, z # kiểm tra các giá trị của x, y, z
(1, 10, 15)
>>>
```

Chúng ta đã biết kiểu của biến nhớ chỉ được xác định tại thời điểm thực hiện lệnh gán.

Trong ví dụ sau, ban đầu `x` và `y` được gán các giá trị nguyên, do đó sẽ có kiểu **int**. Nhưng ở lần sau, `y` được gán một phép tính thực, và do đó `y` sẽ nhận kiểu giá trị **float**.

```
>>> x = 5
>>> y = x + 1
>>> type(x), type(y)
(<class 'int'>, <class 'int'>)
>>> y = x // 1.2
>>> type(y)
<class 'float'>
>>>
```

c. Gán đồng thời

Nếu các biến nhớ có giá trị bằng nhau thì có thể gán bằng một lệnh như sau:

```
>>> x = y = z = 12
>>>
```

Việc gán đồng thời nhiều biến nhớ với các giá trị khác nhau có thể thực hiện như sau:

```
>>> x, y, z = 1, 2, 3
```

```
>>>
```

Chú ý: không được viết như sau, ví dụ: $x = 1, y = 2, z = 3$.

```
>>> x = 1, y = 2, z = 3
```

```
SyntaxError: can't assign to literal
```

```
>>>
```

Phép gán đồng thời trong Python có rất nhiều ý nghĩa. Ví dụ chúng ta muốn đổi giá trị 2 biến nhớ x, y .

Với các ngôn ngữ bậc cao khác, việc đổi giá trị của 2 biến nhớ x, y thường được thực hiện như sau, sử dụng thêm 1 biến nhớ trung gian temp.

```
>>> x, y = 1, 7
```

```
>>> x, y
```

```
(1, 7)
```

```
>>> temp = x
```

```
>>> x = y
```

```
>>> y = temp
```

```
>>> x, y
```

```
(7, 1)
```

```
>>>
```

Tuy nhiên với Python và lệnh gán đồng thời các biến nhớ, việc đổi giá trị 2 biến nhớ x, y được thực hiện chỉ bằng một lệnh như sau.

```
x, y = y, x
```

Ví dụ:

```
>>> x = 1; y = 7
```

```
>>> x, y
```

```
(1, 7)
```

```
>>> x, y = y, x # đổi giá trị 2 biến x, y
```

```
>>> x, y
```

```
(7, 1)
```

```
>>>
```

d. Phép gán thay đổi

Phép gán thay đổi là lệnh có dạng:

<biến nhớ> <phép toán> = <giá trị>

Chức năng của lệnh: thay đổi <biến nhớ> một giá trị bằng <giá trị> bởi phép toán <phép toán>.

Ví dụ:

```
>>> x += 1 # tăng x lên 1 đơn vị, tương đương x = x + 1
```

```
>>> y -= x # tương đương y = y - x
```

```
>>>
```

Chú ý:

Tạm thời chúng ta hiểu, ví dụ, lệnh $x += 1$ tương đương với lệnh $x = x + 1$. Nhưng trong tương lai chúng ta sẽ biết 2 lệnh đó về bản chất là khác nhau.

4. Từ khóa trong Python

Từ khóa là các từ mà Python đã dùng cho các mục đích riêng của ngôn ngữ hoặc dành cho các hàm hệ thống. Vì vậy khi đặt tên biến nhớ trong Python cần tránh đặt trùng với các từ khóa này.

Danh sách từ khóa của Python được cập nhật và mở rộng theo các lần phát triển của mình.

Bảng các từ khóa của Python 3.

False	None	True	and	as	assert	break
class	continue	def	del	elif	else	except
finally	for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass	raise
return	try	while	with	yield		

Chú ý riêng cho biến nhớ "`_`": Ký tự `_` (underscore) sẽ được hiểu là biến nhớ đặc biệt. Biến nhớ này không cần gán mà mặc định luôn được gán giá trị của biểu thức được tính gần đây nhất.

Ví dụ:

```
>>> 12 + 7
19
>>> _
19
>>>
```

5. Các phép tính nâng cao khác

a. Hàm `divmod()`

Chúng ta đã biết trong phần trên về ý nghĩa các phép tính `//` và `%` trong Python.

`//` là phép chia nguyên. Phép tính này trong một số ngôn ngữ lập trình khác ký hiệu là `div`. `%` là phép tính lấy số dư. Phép tính này trong một số ngôn ngữ lập trình khác ký hiệu là `mod`.

Thực chất trong Python thì:

`//` là phép chia làm tròn xuống, hay có thể dùng công thức `// = floor(/)`.

`%` là phép tính lấy số dư chính xác.

Xét các ví dụ sau để hiểu rõ điều trên.

```
>>> 12.5 // 2
6.0
>>> 12.5 % 2
0.5
>>>
```

Hàm `divmod()` trong Python có thể dùng để thay thế cho cả 2 phép tính `//` và `%`. Có thể hiểu công thức tính của hàm `divmod()` như sau:

$$\text{divmod}(x, y) = x // y, x \% y.$$

Ví dụ:

```
>>> divmod(12.5, 2)
(6.0, 0.5)
>>> divmod(12, 3)
(4, 0)
>>>
```

Như vậy nếu viết:

```
>>> divmod(x, y)
```

là tương đương với viết lệnh:

```
>>> x//y, x%y
```

b. Phép tính mũ và số lớn

Chúng ta đã biết phép tính lấy số mũ `**` trong Python được mặc định tính từ phải sang trái. Ví dụ:

$$2 ** 3 ** 2 = 2 ** (3 ** 2).$$

```
>>> 2**3**2
512
>>> (2**3)**2
64
>>>
```

Phép tính số mũ này cũng có thể áp dụng cho số thực.

```
>>> -2.0 ** 10
-1024.0
>>> 3.56 ** 12.75
10739584.369722784
>>>
```

Chú ý: các số lớn trong Python có thể viết theo kiểu số mũ như sau:

$$12e50 = 12.10^{50}$$
$$1.56e-2 = 1.56.10^{-2}$$

Khi một phép tính tạo ra số thực quá lớn, Python sẽ tự động chuyển sang dạng mô tả như trên.

```
>>> 3.5 ** 500
1.0814891708785585e+272
>>>
```

c. Viết lệnh trên nhiều dòng

Mặc định mỗi lệnh của Python chỉ được phép viết trên một dòng. Tuy nhiên có nhiều lúc do lệnh, biểu thức quá dài hoặc có nhu cầu viết xuống dòng cho đẹp thì có thể thực hiện theo các cách sau.

Có thể ngắt dòng, biểu thức đang viết bằng ký tự \, sau đó ENTER xuống dòng và viết tiếp lệnh, biểu thức như bình thường. Python sẽ hiểu là vẫn trên một dòng lệnh.

Ví dụ:

```
>>> x = 1 + 2 + 3 \
      + 4
>>> x
10
```

Các biểu thức nếu có ngoặc (,) thì giữa hai ngoặc này có thể xuống dòng bằng cách gõ ENTER, sau đó viết bình thường.

Ví dụ:

```
>>> x = 12 + (3 - 6**7
          + 12 - (3 + 4))
>>> x
-279916
>>>
```

Khi nhập một chuỗi ký tự, nếu chuỗi đó quá dài thì có thể viết xuống dòng mà vẫn giữ được nguyên cả chuỗi bằng cách sử dụng 3 dấu " hoặc ' để khép lại chuỗi ký tự đó.

Ví dụ muốn viết một câu thơ như sau trên màn hình:

```
Dù ai nói ngà nói nghiêng
Lòng ta vẫn vững như kiềng ba chân.
```

Chúng ta có thể thực hiện lệnh như sau, chú ý sau chữ "nghiêng" cần ENTER để xuống dòng.

```
>>> x = """ Dù ai nói ngà nói nghiêng
Lòng ta vẫn vững như kiềng ba chân."""
>>> print(x)
Dù ai nói ngà nói nghiêng
Lòng ta vẫn vững như kiềng ba chân.
>>>
```

6. Namespace

Chúng ta sẽ tìm hiểu sâu hơn một chút các lệnh gán trong Python để cảm nhận sự khác biệt so với các ngôn ngữ lập trình khác.

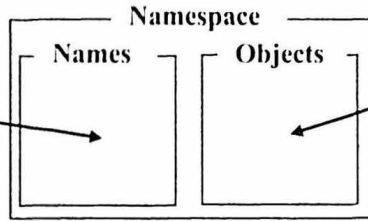
Câu hỏi đặt ra là, một lệnh gán, ví dụ `x = 5`, được thực hiện như thế nào?

Khi lệnh `x = 5` được thực hiện, theo mô hình của các ngôn ngữ lập trình khác là chương trình sẽ dành một vùng bộ nhớ cho biến `x` và sau đó đặt giá trị 5 vào vùng bộ nhớ này.

Python điều khiển hoàn toàn khác.

Trong Python, tất cả các giá trị dữ liệu bao gồm số, chuỗi ký tự, logic đều là các đối tượng (object) được quản lý theo đúng nghĩa (chúng ta sẽ tìm hiểu sau các đối tượng này). Mỗi biến nhớ có một tên riêng của mình và được Python quản lý cùng với các đối tượng số trong một không gian được gọi là **Namespace** (không gian tên). Không gian tên này sẽ điều khiển cả 2 loại đối tượng là các **Names** (biến nhớ) và **Objects** (đối tượng).

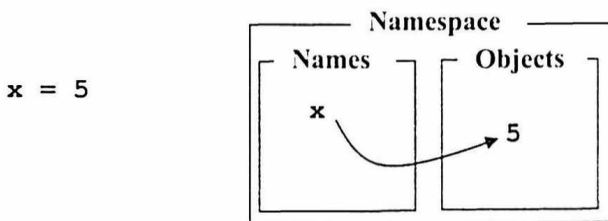
Khu vực quản lý các **tên biến nhớ**. Mỗi biến nhớ khi thực hiện lệnh gán sẽ được kết nối với một đối tượng giá trị ở bên kia.



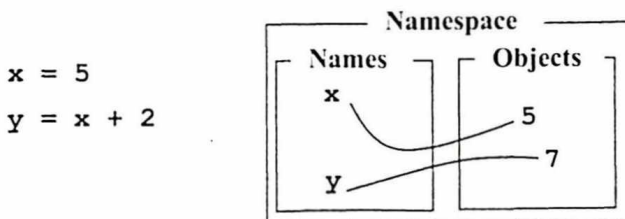
Khu vực quản lý các **đối tượng** giá trị, mỗi giá trị là một đối tượng. Đối tượng giá trị có thể là số, xâu ký tự, logic, ...

Trong không gian **Namespace** này, mỗi khi thực hiện lệnh gán, ví dụ: `x = 5`, Python sẽ tính toán giá trị cần gán trong vùng **Objects**, sau đó tạo liên kết giữa biến nhớ `x` với giá trị số này (ví dụ 5). Giá trị số này được gọi là **lvalue**.

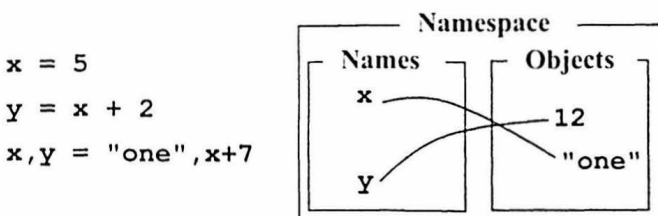
Hình sau mô tả cách Python thực hiện lệnh `x = 5`. Đối tượng số sẽ tính giá trị `lvalue = 5` và tạo liên kết giữa `x` và `lvalue` này. Như vậy `x` chỉ hình thành tại thời điểm gán.



Bây giờ giả sử chúng ta thực hiện tiếp lệnh `y = x + 2`. Khi đó biểu thức bên phải của lệnh gán được tính toán trong vùng **Objects**, giá trị tính được là 7 sẽ chính là `lvalue = 7`. Sau đó Python thực hiện kết nối biến nhớ `y` với giá trị này, tức là `y = 7`. Giá trị biến `x` không thay đổi.



Tiếp theo khi thực hiện lệnh `x,y = "one", x+7` thì đồng thời vùng **Objects** sẽ tính toán, tạo ra "one" và giá trị 12 (`= x + 7`), sau đó sẽ tạo kết nối đồng thời `x → "one"` và `y → 12`.



Dãy các lệnh trên được thực hiện trên Python Shell như sau.

```
>>> x = 5
>>> y = x + 2
>>> x, y = "one", x + 7
>>> x, y
('one', 12)
>>>
```



Câu hỏi - bài tập

1. Em hãy giải thích vì sao một biến nhớ trong Python tại các thời điểm khác nhau có thể gán các giá trị dữ liệu có kiểu khác nhau.

2. Các lệnh sau đây có hợp lệ không?

A. $x = y = z = 1$

B. $x = 1, y = 2, z = 3$

C. $x, y = 1$

D. $x, y, z = 1, 2, 3$

3. Sau các lệnh:

```
>>> x = y = 1
>>> x = x + 1
```

thì giá trị y bằng bao nhiêu?

4. Sau các lệnh:

```
>>> x = 10
>>> y = x
>>> x = 20
```

thì giá trị y bằng bao nhiêu?

5. Sau các lệnh:

```
>>> x = y
>>> y = x
```

Thì giá trị các biến nhớ x, y thay đổi như thế nào so với trước khi thực hiện các lệnh này?

6. Có thể đặt tên biến nhớ trùng với từ khóa của Python được hay không?

7. Lệnh

```
>>> print(13 + 25)
```

Sẽ in ra kết quả là gì?

a) $13 + 25$

b) $'13 + 25'$

c) 38

d) $'38'$

8. Đặt tên biến nhớ như sau có được không?

a) a-b-c

b) a_b_c

c) __abc__

d) -a_b_c-

9. Các tên định danh sau có hợp lý không trong Python?

a) l_abc

b) A.B.C

c) ____l_

d) l.abc

e) a&b&c

10. Đặt tên các biến nhớ như sau có hợp lệ không?

a) None

b) none

c) Việt Nam

d) x y z

e) break

f) happy-day

11. Hai biểu thức $x ** 2$ và $x * 2$ có đồng nhất giá trị không với các giá trị khác nhau của x là nguyên hay thập phân?

12. Hai biểu thức $x ** 2.0$ và $x * x$ có đồng nhất giá trị không với các giá trị khác nhau của x là nguyên hay thập phân?

13. Kết quả của lệnh, biểu thức `print(1 - 4 % 6)` như thế nào?

a) $1 - 4 \% 6$

b) -3

c) 3

14. Biểu thức `print(- 4 % 6)` có kết quả như thế nào?

a) $- 4 \% 6$

b) - 4

c) 2

15. Hai biểu thức $(2 ** 3) ** 4$ và $2 ** 3 ** 4$ có giá trị giống nhau hay khác nhau?

16. Hãy kiểm tra xem công thức:

$$a = b * (a // b) + (a \% b).$$

có còn đúng không nếu a, b không là số nguyên.

17. Nếu a, b đồng thời là số nguyên thì giá trị các biểu thức a / b và $a // b$ có đồng nhất với nhau không?

18. Giả sử ss = giá trị tính bằng số giây cho trước. Khi đó muốn đổi giá trị này ra phút và giờ thì dùng các công thức tính nào? Chú ý chỉ cần tính tròn phút và tròn giờ.

19. Giả sử có đoạn chương trình sau:

```
>>> a = 5
>>> b = 7
>>> a, b = b, a
```

Sau các lệnh trên giá trị các biến nhớ a, b bằng bao nhiêu?

a) a = 5, b = 5

b) a = 5, b = 7

c) a = 7, b = 7

d) a = 7, b = 5

20. Công thức tính theo radian (rad) của một góc tính bằng độ (angle) như sau:

$\text{Rad} = \text{angle} \times 3.14 / 180.$

Viết lệnh bằng Python mô tả công thức trên.

21. Sau các lệnh sau thì x, y nhận các giá trị nào?

```
>>> x = 2
>>> y = 5
>>> x += 1
>>> x, y = y, x
>>> y += 1
```

22. Trên thế giới có 2 cách thể hiện nhiệt độ: Nhiệt độ C theo Celsius và nhiệt độ K theo Kenvin. Công thức tính nhiệt độ T theo Kenvin tính theo nhiệt độ t theo Celsius như sau:

$T = t + 273.$

Viết công thức trong Python tính nhiệt độ Kenvin theo Celsius và ngược lại.

23. Có thể tính biểu thức $1 + 2 + 3 + \dots + 10$ chỉ bằng 4 lệnh trong Python hay không, trong đó tất cả các biểu thức tính toán của các lệnh này có không quá 2 phép cộng?

24. Dùng mô hình biểu diễn **Namespace** để giải thích kết quả lệnh:

$x, y = y, x$

25. Cho trước các giá trị s tính bằng giây, m tính bằng phút, h tính bằng giờ, d tính bằng ngày. Viết công thức đổi giá trị "d ngày, h giờ, m phút, s giây" ra:

a) giây.

b) phút.

c) giờ.

Chương 3. Input và chuyển đổi dữ liệu

Mục tiêu:

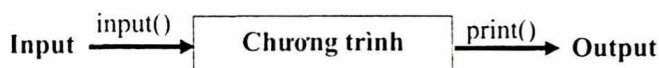
- Tìm hiểu chức năng lệnh `input()` và chuyển đổi trực tiếp kiểu dữ liệu trong Python.
- Bước đầu làm quen với kiểu dữ liệu `bool`, lệnh `if`.
- Bước đầu làm quen với lệnh `for`, `while`.



Nội dung chính

1. Input dữ liệu

Trong mọi hệ thống đều cần có sự tương tác trao đổi dữ liệu với bên ngoài. Sự tương tác đó thông qua các hệ thống Input, Output như hình dưới đây. Chúng ta đã được làm quen với một lệnh Output đầu tiên của Python là lệnh `print()`. Bây giờ chúng ta sẽ làm quen với lệnh `Input()`.



Hàm `input()` dùng để thực hiện việc nhập dữ liệu từ bàn phím, dữ liệu được nhập chính là giá trị trả lại của hàm số này.

Hàm `input()` có cú pháp như sau:

```
<biến nhớ> = input(<dòng chữ hiện trên màn hình>)
```

Khi thực hiện lệnh <dòng chữ hiện trên màn hình> sẽ xuất hiện, người dùng nhập một xâu ký tự bất kỳ, kết thúc bằng bấm ENTER. Kết quả của hàm sẽ lưu vào <biến nhớ>.

Xét ví dụ đơn giản sau.

```
>>> Name = input("Hãy nhập họ tên: ")
Hãy nhập họ tên: Bùi Việt Hà
>>> print("Xin chào", Name)
Xin chào Bùi Việt Hà
>>>
```

Chương trình `Xinchao.py` mô tả cách thực hiện lệnh `input()` thông qua môi trường tương tác Python.

Thiết lập tệp `Xinchao.py` có nội dung như sau:

```
1. # thử lệnh input()
2. Name = input("Hãy nhập họ tên: ")
3. print("Xin chào", Name)
```

Chương trình trên chạy sẽ có kết quả, ví dụ, như hình sau.

Debug I/O (stdin, stdout, stderr) appears below

```
Hãy nhập họ tên: Bùi Việt Hà
Xin chào Bùi Việt Hà
```


Chú ý rằng hàm `input()` luôn trả lại dữ liệu kiểu chuỗi ký tự, mặc dù trên màn hình dữ liệu thể hiện như số bình thường.

Trong ví dụ sau dữ liệu nhập từ bàn phím vào biến nhớ `num` là 30, cụ thể `num` sẽ là chuỗi "30" chứ không phải là số 30.

```
>>> num = input("Hãy nhập 1 số: ")
Hãy nhập 1 số: 30
>>> type(num)
<class 'str'>
>>>
```

2. Chuyển đổi trực tiếp dữ liệu

Để có thể nhập được chính xác kiểu dữ liệu muốn nhập, chúng ta sẽ sử dụng các hàm chuyển đổi trực tiếp dữ liệu đơn giản của Python. Đó là các hàm `int()`, `float()` và `str()`.

a. Hàm `int()`

Hàm `int()` có cú pháp là:

<giá trị trả về> = `int`(<giá trị>)

Hàm này sẽ trả lại giá trị là số nguyên là chuyển đổi của tham số <giá trị>. Hàm sẽ chỉ có tác dụng khi <giá trị> là một số thực hoặc là chuỗi ký tự của một số nguyên. Nếu đối số là số thực thì hàm `int()` sẽ lấy ra phần nguyên của số này.

Ví dụ `int(2.56) = 2`, `int(-15.09) = -15`.

Chú ý đến các lệnh thứ 3, 4 trong ví dụ sau để thấy hàm `int()` không thể chuyển đổi chuỗi gốc là số thực hay biểu thức số.

```
>>> int("23")
23
>>> int(23.45)
23
>>> int("12.55")
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    int("12.55")
ValueError: invalid literal for int() with base 10: '12.55'
>>> int("34 + 67")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    int("34 + 67")
ValueError: invalid literal for int() with base 10: '34 + 67'
>>>
```

b. Hàm `float()`

Hàm `float()` có cú pháp là:

<giá trị trả về> = `float`(<giá trị>)

Hàm này sẽ trả lại giá trị là số thực là chuyển đổi của tham số <giá trị>. Hàm sẽ chỉ có tác dụng khi <giá trị> là một biểu thức số bất kỳ hoặc là chuỗi ký tự của một số thực hoặc số nguyên.

Chú ý: Hàm sẽ báo lỗi khi <giá trị> là một xâu chứa biểu thức số bất kỳ.

Trong các ví dụ sau, chú ý đến lệnh chuyển đổi cuối cùng bị lỗi do hàm float() không chuyển đổi được xâu ký tự của một biểu thức.

```
>>> float("3.14156")
3.14156
>>> float(5)
5.0
>>> float(14 + 67 + 3)
84.0
>>> float("13 + 7.3")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    float("13 + 7.3")
ValueError: could not convert string to float: '13 + 7.3'
>>>
```

Để xử lý xâu biểu thức xem hàm eval() ở phần sau.

Muốn chuyển đổi một xâu của số thực chúng ta sẽ cần sử dụng 2 hàm int() và float() lồng nhau như ví dụ sau.

```
>>> int(float("3.141567"))
3
>>>
```

c. Hàm str()

Hàm str() có chức năng chuyển đổi một giá trị hoặc một biểu thức số sang kiểu xâu ký tự. Cú pháp tương tự như 2 hàm int() và float() đã biết.

<giá trị trả về> = str(<biểu thức>)

Hàm này sẽ chuyển đổi một biểu thức số bất kỳ sang dạng xâu. Nếu <biểu thức> đã là xâu rồi thì hàm sẽ trả lại giá trị gốc của xâu này. Chú ý rằng <biểu thức> có thể phức tạp bất kỳ. Xét một số ví dụ sau.

```
>>> str(5)
'5'
>>> str(3 + 4**5 - 17.8**2)
'710.16'
>>>
```

Chú ý:

Hàm str() không hỗ trợ chuyển đổi một dãy dữ liệu, ví dụ khi viết (3, 4) nhưng lại hỗ trợ chuyển đổi kết quả của hàm divmod(). Xem các ví dụ sau:

```
>>> str(3,4)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    str(3,4)
TypeError: str() argument 2 must be str, not int
>>> str(divmod(19,5))
'(3, 4)'
>>>
```

d. Một số ví dụ

Ví dụ 1: Chương trình nhập lần lượt 2 số tự nhiên m, n từ bàn phím.

Nhap hai so tu nhien.py.

```
1. # Nhập lần lượt 2 số m, n từ bàn phím.  
3. m = int(input("Nhập số m: "))  
4. n = int(input("Nhập số n: "))  
5. print("2 số m, n đã nhập là m =", m, "n =", n)
```

Khi thực hiện chương trình này, kết quả thể hiện trên màn hình tương tác sẽ như sau.

```
Nhập số m: 11  
Nhập số n: 8  
2 số m, n đã nhập là m = 11 n = 8
```

Ví dụ 2: Chương trình yêu cầu nhập tên, sau đó là tuổi của người sử dụng.

Nhap ten va tuoi.py.

```
1. # Nhập lần lượt tên và tuổi của người dùng.  
2. name = input("Nhập họ tên của bạn: ")  
3. n = int(input("Xin chào bạn " + name + ". Bạn năm nay bao nhiêu  
   tuổi: "))  
4. print("Chúc bạn đã nhập xong dữ liệu.")
```

3. Hàm eval() tính toán giá trị từ xâu biểu thức

Chúng ta đã biết được ý nghĩa của 3 hàm số int(), float() và str() dùng để chuyển đổi trực tiếp các kiểu dữ liệu cơ bản, được dùng nhiều trong quá trình nhập liệu bằng lệnh input(). Tuy nhiên còn một khó khăn nữa mà các lệnh trên không thể chuyển đổi được khi làm việc với các xâu ký tự là biểu thức toán học. Ví dụ các xâu như "12 + 5" hay "23 ** 2 - 12" sẽ không thể hiểu được bởi các hàm int() hay float().

Hàm số eval() chính là hàm chúng ta cần trong các trường hợp này.

Cú pháp của hàm eval().

<giá trị trả về> = eval("<biểu thức>")

Hàm số này có tác dụng tính toán các biểu thức bên trong một xâu ký tự và chuyển đổi chúng về dạng số tương ứng.

Xét một vài ví dụ ban đầu.

```
>>> eval("12 + 7")  
19  
>>> eval("24 ** 2 - 15")  
561  
>>> eval("15**2 + 7.0 + 13*(34 - 2**1.5)")  
637.2304473782995
```

Ví dụ chương trình sau yêu cầu người dùng nhập một biểu thức toán học từ bàn phím, sau khi nhập, chương trình sẽ tính toán và đưa ra ngay kết quả.

Nhap biểu thức.py.

```
1. # Nhập biểu thức từ bàn phím.  
2. x = eval(input("Nhập biểu thức toán học bất kỳ: "))  
3. print("Giá trị biểu thức đã nhập là:", x)
```

Kết quả của chương trình trên có thể như sau:

```
Nhập biểu thức toán học bất kỳ: 2**2**3 + 320  
Giá trị biểu thức đã nhập là: 576
```

Một tính năng rất đặc biệt nữa là hàm `eval()` có thể nhận biết đồng thời nhiều biểu thức trên một hàng, cách nhau bởi dấu phẩy.

Ví dụ:

```
>>> eval("1+2, 3**2, 14.2 + 16.5**2")  
(3, 9, 286.45)  
>>> m,n = eval("3, 4")  
>>> m, n  
(3, 4)
```

Tính năng này của hàm `eval()` có rất nhiều ứng dụng trên thực tế.

Ví dụ: Nhập nhiều số tự nhiên trên một dòng từ bàn phím.

Nhap 3 so.py.

```
1. # Nhập 2 hoặc nhiều số trên một hàng.  
2. m,n,p=eval(input("Nhập các số m,n,p (cách nhau bởi dấu phẩy):  
   "))  
3. print("Các số đã nhập là:",m,n,p)
```

4. Kiểu bool và lệnh if¹

Trong phần này chúng ta sẽ tìm hiểu sâu hơn về kiểu dữ liệu bool (logic) và lệnh điều khiển rẽ nhánh `if`. Với lệnh `if` chúng ta sẽ lần đầu tiên làm quen với nguyên tắc viết lệnh "thụt vào" (indent) của Python.

a. Kiểu dữ liệu bool (logic)

Dữ liệu kiểu bool (logic) là loại dữ liệu chỉ có 2 giá trị Đúng (True) và Sai (False). Dữ liệu bool được dùng khi mô tả các điều kiện hoặc phép so sánh số hoặc chữ.

Trong Python, 2 giá trị chính của biểu dữ liệu này là **True** và **False**. Hàm `bool()` sẽ kiểm tra xem giá trị của đối số của hàm là True hay False.

Cùng xem một số ví dụ sau:

```
>>> 3 > 2  
True  
>>> x = 2 > 3  
>>> bool(x)  
False  
>>>
```

¹ Trong bài học này chúng ta sẽ tìm hiểu nhanh các lệnh `if`, `for` và `while` để bước đầu có thể giải các bài tập ngắn và nhanh. Trong các bài học tiếp theo các lệnh này sẽ được giới thiệu đầy đủ hơn.

Ở lệnh đầu tiên, khi gõ $3 > 2$, đó chính là một biểu thức logic với phép so sánh, biểu thức này có giá trị bool là True (đúng).

Ở lệnh thứ hai, phép gán $x = 2 > 3$, biểu thức logic $2 > 3$ nhận giá trị False được gán cho biến nhớ x và x sẽ nhận giá trị False (sai).

Phép so sánh số là cách đơn giản nhất để tạo ra các kiểu dữ liệu logic này. Bảng sau mô tả qui tắc ghi các phép so sánh số trong Python.

Toán tử so sánh	Ý nghĩa	Câu hỏi so sánh
$x == y$	Phép so sánh bằng nhau.	x có bằng y hay không?
$x != y$	Phép so sánh không bằng nhau.	x có khác y hay không?
$x > y$	Phép so sánh lớn hơn.	x có lớn hơn y hay không?
$x >= y$	Phép so sánh lớn hơn hoặc bằng.	x có lớn hơn hoặc bằng y hay không?
$x < y$	Phép so sánh nhỏ hơn.	x có nhỏ hơn y hay không?
$x <= y$	Phép so sánh nhỏ hơn hoặc bằng.	x có nhỏ hơn hoặc bằng y hay không?

Có 3 phép toán chính với kiểu dữ liệu bool là and (và), or (hoặc) và not (phủ định). Bảng giá trị của các phép toán này được định nghĩa như sau.

Phép toán và **and**

x	and	y	x and y
True	and	True	True
False	and	False	False
True	and	False	False
False	and	True	False

Phép toán hoặc **or**

x	or	y	x or y
True	or	True	True
False	or	False	False
True	or	False	True
False	or	True	True

Phép toán phủ định **not**

x	not x
True	False
False	True

b. Lệnh if

Lệnh điều khiển rẽ nhánh **if** là một trong những lệnh được dùng nhiều nhất trong mọi ngôn ngữ lập trình. Ý nghĩa thực tế của lệnh có thể hiểu và hình dung như trong câu nói tình huống sau:

Nếu sáng nay trời tạnh thì tôi sẽ đi chơi, còn nếu không tôi sẽ ở nhà học bài.

Trong mệnh đề phát biểu trên có 2 hành động là "đi chơi" và "học bài". Việc thực hiện các hành động đó không phải tuần tự, mà phụ thuộc vào điều kiện của thời tiết sau từ khóa "nếu". Trong lập trình các lệnh được thực hiện theo điều kiện như vậy sẽ được thể hiện bằng lệnh **if**.

Lệnh này mô tả bằng ngôn ngữ con người như sau:

Nếu <điều kiện đúng> thì
<thực hiện các nhóm lệnh này>

Lệnh **if** nói chung và trong Python nói riêng có 2 dạng, dạng ngắn và dạng đầy đủ như sau.

Cấu trúc lệnh **if**, cú pháp ngắn:

Chú ý sau <điều kiện> cần có dấu ":".

if <điều kiện kiểm tra>:
<các lệnh của nhóm if>

Toàn bộ cụm các lệnh của nhóm **if** phải viết thụt vào (tối thiểu một dấu cách) so với **if**.

Chú ý: tất cả các lệnh trong nhóm **if** này đều phải **thụt vào như nhau**. Điều này rất quan trọng.

Lệnh trên sẽ thực hiện như sau:

Chương trình sẽ kiểm tra <điều kiện> sau từ khóa **if**, nếu điều kiện là đúng thì nhóm <các lệnh của nhóm if> sẽ thực hiện. Nếu điều kiện sai thì lệnh sẽ được bỏ qua.

Cấu trúc lệnh **if**, cú pháp đầy đủ:

if <điều kiện kiểm tra>:
<các lệnh của nhóm if>
else:
<các lệnh của nhóm else>

Chú ý sau <điều kiện> của **if** và sau **else** cần có dấu ":".

Toàn bộ cụm <các lệnh của nhóm **if**> và <các lệnh của nhóm **else**> phải viết thụt vào (tối thiểu một dấu cách) so với **if** và **else**.

Chú ý: tất cả các lệnh trong nhóm **if** và **else** này đều phải **thụt vào như nhau**.

Lệnh trên sẽ thực hiện như sau:

Chương trình sẽ kiểm tra <điều kiện> sau từ khóa **if**, nếu điều kiện là đúng thì nhóm <các lệnh của nhóm if> sẽ thực hiện. Nếu điều kiện sai thì <các lệnh của nhóm else> sẽ được thực hiện.

Xét một số ví dụ sau đây.

Lệnh sau <điều kiện> là đúng nên lệnh của nhóm **if** sẽ luôn được thực hiện.

```
>>> if True:
    print("Tôi đi chơi đây")
Tôi đi chơi đây
>>>
```

Ví dụ tiếp theo sẽ là ngược lại do <điều kiện> sai nên lệnh sẽ không được thực hiện.

```
>>> t = 2
>>> if t > 2:
    print(t, "có giá trị lớn hơn 2")
```

```
>>>
```

Chương trình Python đơn giản sau sẽ yêu cầu người dùng nhập vào 2 số nguyên m, n, sau đó thông báo các số này bằng nhau hay số nào lớn hơn. Chúng ta chú ý cách sử dụng lệnh if lồng nhau như thế nào.

Nhap so sanh 2 so.py.

```
1. # Nhập 2 hoặc nhiều số trên một hàng.
2. m,n=eval(input("Nhập các số m,n (cách nhau bởi dấu phẩy): "))
3. if m==n:
4.     print("Hai số m, n bằng nhau")
5. else:
6.     if n>m:
7.         print("Số n lớn hơn m")
8.     else:
9.         print("Số m lớn hơn n")
```

Trong chương trình trên chú ý đến lệnh if - else tại dòng 6. Lệnh này nằm trong nhóm các lệnh else của lệnh if tại dòng 3 phía trên.

5. Lệnh lặp for

Những người lập trình theo các ngôn ngữ bậc cao cổ điển (như C, C++, Pascal, Java) đều đã quá quen thuộc với lệnh for. một lệnh lặp với số bước hữu hạn rất hay được sử dụng. Trong Python cũng có lệnh for như vậy. Trong bài học này chúng ta cùng nhau tìm hiểu tính năng của lệnh for trong Python tương tự như trong các ngôn ngữ lập trình cổ điển khác.

Bảng sau cho chúng ta thấy sự tương đồng về ý nghĩa và cách viết lệnh for trong C++ và Python.

C++	Python	Mô tả ý nghĩa
for (k =1, k<100, k++) {<nhóm lệnh for>}	for k in range(1,100): <nhóm lệnh for>	Lần lượt cho biến nhớ k gán các giá trị tăng từ 1 đến 99, mỗi lần như vậy thực hiện lặp <nhóm lệnh for>
for (k=100, k>0, k--) {<nhóm lệnh for>}	for k in range(100,0,-1): <nhóm lệnh for>	Lần lượt cho biến nhớ k gán các giá trị giảm từ 100 xuống đến 1, mỗi lần như vậy thực hiện lặp <nhóm lệnh for>

Chú ý: Trong Python chúng ta gặp lại cấu trúc quen thuộc của lệnh for tương tự như lệnh if đã nói ở phần trên. Sau câu lệnh chính của for cần có dấu ":", sau đó xuống dòng và <nhóm các lệnh for> bắt buộc phải viết thụt vào như nhau. Như vậy trong Python, các dấu cách thụt vào đầu dòng của lệnh đóng vai trò rất quan trọng.

Hàm hay biểu thức **range()** sẽ tạo ra một dãy giá trị có thể đếm được trong Python. Hàm này có nhiều dạng, có thể mô tả tóm tắt trong bảng sau.

Cú pháp tổng quát: **range(<start>,<end>,<increment>)**

Cú pháp của range()	Ý nghĩa
range(0, n)	Dãy các giá trị từ 0, 1, ... đến n-1.
range(1, n, 2)	Dãy các giá trị từ 1, 3, ... đến n-1, tăng dần theo 2 đơn vị.
range(n)	Tương đương range(0, n, 1) - Dãy các giá trị từ 0, 1, ... đến n-1.
range(n, 0, -1)	Dãy các giá trị n, n-1, ..., 1 (giảm dần).

Chúng ta cùng xét một vài ví dụ để hiểu rõ hơn lệnh lặp **for**.

1. Cho trước số tự nhiên n, cần tính tổng $S = 1 + 2 + \dots + n$. Công việc tính toán sẽ thực hiện trực tiếp trên giao diện tương tác của Python Shell.

```
>>> n= 100
>>> S = 0
>>> for k in range(1,n+1):
        S = S + k
```

```
>>> S
5050
>>>
```

Với n gán = 100, giá trị S được tính là 5050.

2. Chương trình sau yêu cầu nhập từ bàn phím số tự nhiên n, sau đó sẽ tính số các ước số thực sự của n.

Dem_so_uoc_so.py.

```
1. n = int(input("Nhập số tự nhiên n: "))
2. count = 0
3. for k in range(1, n):
4.     if n % k == 0:
5.         count +=1
6. print("Số các ước số thực sự của",n,"là:",count)
```

6. Lệnh lặp while

Một lệnh nữa cần đưa ra ở đây là lệnh lặp theo điều kiện **while**. Khác với lệnh **for** là lặp theo số lần lặp cố định, lệnh **while** có số vòng lặp không cố định mà phụ thuộc vào điều kiện dừng của lệnh.

Cú pháp đơn giản như sau.

```
while <điều kiện>:
    <các lệnh lặp>
```

Chú ý có dấu ":", sau đó các lệnh lặp cần viết thụt vào.

Các lệnh tại đây cần viết thụt vào.

Lệnh này sẽ chạy liên tục <các lệnh lặp> trong thân của **while** và chỉ dừng theo <điều kiện>. Mỗi lần chạy <các lệnh lặp>, chương trình sẽ kiểm tra <điều kiện>, nếu đúng thì chạy tiếp, nếu sai thì dừng lại, kết thúc lệnh.

Ví dụ: Nhập 2 số tự nhiên m, n từ bàn phím và tính USCLN của 2 số này.

USCLN.py

```
1. M = int(input("Nhập số tự nhiên m: "))
2. N = int(input("Nhập số tự nhiên n: "))
3. m = M; n = N
4. while n != m:
5.     if n > m:
6.         n = n - m
7.     else:
8.         m = m - n
9. print("USCLN của", M, "và", N, "là:", n)
```

Kết quả khi chạy chương trình trên có dạng như sau.

```
Nhập số tự nhiên m: 15
Nhập số tự nhiên n: 12
USCLN của 15 và 12 là: 3
```

Chú ý: Trong lệnh lặp **while**, <điều kiện> sẽ được kiểm tra trước khi thực hiện các lệnh trong vòng lặp. Do đó nếu <điều kiện> bị sai ngay từ đầu thì sẽ không có lệnh nào được thực hiện.



Câu hỏi - bài tập

1. Các biểu thức sau sẽ có giá trị đúng hay sai?

- a) $1 < 2 == 3 < 4$
- b) $4 < 4 - 1$
- c) `False == False`
- d) `True == False`

2. Giả sử thực hiện dòng lệnh sau trên Python:

```
x = input("Enter x: ")
```

Khi thực hiện người dùng sẽ nhập từ dòng yêu cầu nhập như sau: $3 + 4 * 5$

Hỏi giá trị x khi đó là gì?

- a) $3 + 4 * 5$
- b) 23
- c) `'3 + 4 * 5'`

3. Giả sử em thực hiện các lệnh sau từ cửa sổ tương tác của Python, sau đó nhập dữ liệu tại chỗ:

```
>>> x = int(input("Nhập x: "))
Nhập x: 3 - 4/7 + 2**3
```


Kết quả sau các lệnh trên x nhận giá trị nào?

a) 10

b) 11

c) Python thông báo lỗi.

4. Giả sử em thực hiện các lệnh sau từ cửa sổ tương tác của Python, sau đó nhập dữ liệu tại chỗ:

```
>>> x = int(input("Nhập x: "))  
Nhập x: 5.0
```

Kết quả sau các lệnh trên x nhận giá trị nào?

a) 5

b) 5.0

c) '5'

d) Python thông báo lỗi.

5. Giả sử em thực hiện các lệnh sau từ cửa sổ tương tác của Python, sau đó nhập dữ liệu tại chỗ:

```
>>> x = float(input("Nhập x: "))  
Nhập x: 4
```

Kết quả sau các lệnh trên x nhận giá trị nào?

a) 4

b) 4.0

c) '4.0'

d) Python thông báo lỗi.

6. Mệnh đề sau đúng hay sai:

Hàm eval() chấp nhận trong tham số của mình tất cả các biểu thức sau:

"2 + 2**5", "3 + 4/3, 5 ** 2", "3.456 + 2.890".

7. Viết chương trình nhập 2 số tự nhiên m, n trên một dòng cách nhau bởi dấu phẩy, sau đó hiển thị tổng và tích 2 số này.

8. Viết chương trình nhập 3 số dương a, b, c trên một dòng cách nhau bởi dấu phẩy, kiểm tra xem 3 số này có tạo thành độ dài của một tam giác được hay không, thể hiện trên màn hình dòng chữ "Có thể tạo thành một tam giác" hoặc "Không thể tạo thành một tam giác".

9. Năm nhuận là năm mà giá trị của nó thỏa mãn điều kiện: Chia hết cho 400 hoặc chia hết cho 4 nhưng không chia hết cho 100. Viết biểu thức logic để kiểm tra giá trị số **year** có là năm nhuận hay không.

10. Viết chương trình nhập 3 số tự nhiên từ bàn phím là **day**, **month** và **year** có ý nghĩa là ngày, tháng, năm tương ứng. Kiểm tra xem bộ dữ liệu đã nhập có hợp lý hay không.

11. Viết chương trình nhập 3 số tự nhiên từ bàn phím là **day**, **month** và **year** có ý nghĩa là ngày, tháng, năm tương ứng. Viết chương trình tính tổng số ngày tính từ đầu năm đó (ngày 1/1) cho đến ngày hiện thời. Chú ý đến yếu tố năm nhuận.

12. Viết chương trình nhập số tự nhiên n từ bàn phím, sau đó kiểm tra xem số n có phải là số nguyên tố hay không, thông báo trên màn hình.

13. Viết chương trình nhập từ bàn phím số tự nhiên ss tính theo giây, sau đó đổi số giây này ra thành giờ, phút, giây và thông báo trên màn hình. Ví dụ nếu nhập vào $ss = 3740$ thì kết quả hiển thị như sau:

3740 giây = 1 giờ 2 phút 20 giây.

14. Viết chương trình nhập số tự nhiên n và in ra trên màn hình tất cả các ước số thực sự của n .

15. Viết chương trình nhập số tự nhiên n và đếm số các ước số thực sự của n . In kết quả ra màn hình.

16. Viết các chương trình nhập số tự nhiên n và tính các giá trị sau:

a) Tính tổng $1 + 2 + \dots + 2n$.

b) Tính tổng các số tự nhiên $< n$ và là số lẻ.

c) Tính tổng các số tự nhiên $< n$ và là số chẵn.

17. Viết các chương trình nhập một số tự nhiên n và tính các giá trị sau:

a) Tính số các ước số nguyên tố khác nhau của n (tính cả n).

b) Tính tổng các ước số thực sự của n (không tính n).

c) Tính tổng $1^2 + 2^2 + 3^2 + \dots + n^2$.

18. Cho trước biến nhớ p . Làm thế nào để biết được biến nhớ p thuộc kiểu int, float hay str?

19. Viết chương trình nhập số tự nhiên n từ bàn phím. Tính và kiểm tra tính đúng đắn của biểu thức sau:

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

20. Viết chương trình nhập số tự nhiên n từ bàn phím. Tính và kiểm tra tính đúng đắn của biểu thức sau:

$$1^3 + 2^3 + 3^3 + \dots + n^3 = (1 + 2 + 3 + \dots + n)^2$$

21. Viết chương trình đếm số các số nguyên tố < 10000 .

22. Viết chương trình nhập số tự nhiên n từ bàn phím và in ra số nguyên tố nhỏ nhất lớn hơn hoặc bằng n .

23. Số tự nhiên được gọi là hoàn hảo nếu số đó bằng tổng các ước số thực sự của mình. Ví dụ số 6 là hoàn hảo vì $6 = 1 + 2 + 3$. Viết chương trình nhập số tự nhiên n từ bàn phím và kiểm tra xem n có phải là số hoàn hảo hay không. Thông báo trên màn hình.

24. Viết chương trình tìm tất cả các số tự nhiên hoàn hảo < 10000 .

25. Viết chương trình in ra tất cả các cặp số nguyên tố cách nhau 2 đơn vị và < 10000 . Ví dụ các cặp số nguyên tố đầu tiên cách nhau 2 đơn vị là (3, 5), (5, 7), (11, 13).

Chương 4. Hàm số

Mục tiêu:

- Khái niệm và định nghĩa hàm số trong Python.
- Hàm void và hàm số có giá trị.
- Các chú ý đặc biệt với tham số và phạm vi của hàm số trong Python.



Nội dung chính

1. Khái niệm hàm số trong Python

Hàm số là một khái niệm trung tâm của Python. Trong bài học này chúng ta sẽ làm quen sâu hơn nữa đến khái niệm này và cách tạo các hàm số mới trong Python. Trước tiên chúng ta cùng xem lại 2 “hàm số” đã biết trong các bài trên là `input()` và `print()`.

Hàm `input()` sẽ yêu cầu nhập một chuỗi ký tự từ bàn phím (từ đầu vào - input chuẩn) và trả lại chính chuỗi ký tự đó. Kiểu kết quả của hàm này là “str”.

```
>>> s = input()
abc
>>> type(s)
<class 'str'>
```

Ngược lại, lệnh (hay hàm số) `print()` sẽ không trả lại các giá trị cụ thể nào. Trong Python vẫn cho phép thực hiện lệnh gán, ví dụ `a = print(3)`, tuy nhiên nếu kiểm tra thì kiểu giá trị của `a` sẽ là `NoneType` như ví dụ sau.

```
>>> a = print(3)
3
>>> type(a)
<class 'NoneType'>
```

Biến nhớ `a` thuộc kiểu “`NoneType`” do vậy sẽ không thể thực hiện bất cứ phép tính nào ngoài lệnh gán.

Ví dụ:

```
>>> a + 1
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a + 1
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
>>>
```

2. Định nghĩa hàm số

Trong Python hỗ trợ định nghĩa cả 2 kiểu hàm số đã nói trong phần trên: hàm không có giá trị (hàm void) và hàm có trả lại giá trị.

Trước tiên chúng ta làm quen với cách tạo các hàm không trả lại giá trị. Các hàm này thường được gọi là hàm **void**. Các hàm void còn được gọi là **thủ tục** trong một số ngôn ngữ lập trình khác.

2.1. Định nghĩa hàm void (thủ tục)

Định nghĩa hàm void theo mẫu sau.

```
def <tên hàm số>(<tham số>):  
    <các lệnh mô tả hàm số>
```

Chú ý:

- Tên hàm số cần tuân thủ quy luật đặt tên trong Python.
- Phần thân của hàm số được viết thụt vào theo đúng quy định của Python. Định nghĩa hàm số phải kết thúc bằng ký hiệu ":".
- Trong thân của hàm số có thể có hoặc không có lệnh **return**. Lệnh **return** có tác dụng kết thúc ngay hàm số.
- Hàm số có thể không có hoặc có các tham số. Có thể có nhiều tham số, viết cách nhau bởi dấu phẩy.

Ví dụ một hàm void đơn giản nhất có thể định nghĩa ngay trên cửa sổ lệnh tương tác của Python.

```
>>> def Welcome():  
    print("Xin chào !")  
...  
...  
>>> Welcome()  
Xin chào !  
>>>
```

← Định nghĩa hàm số ngay trên dòng lệnh.

← Gõ và chạy hàm số ngay từ Python Shell tương tác.

Hàm void khi đã định nghĩa sẽ được sử dụng như các lệnh thông thường.

Ví dụ sau là một chương trình Python hoàn chỉnh. Chương trình được cấu trúc bởi 2 thủ tục hay hàm số void. Hàm số thứ 2 (InputData()) có lệnh **return** thông báo kết thúc nhập liệu.

InputData.py

```
1. def Welcome():  
2.     print("Xin chào !")  
3. def InputData():  
4.     m = int(input("Nhập số m: "))  
5.     n = int(input("Nhập số n: "))  
6.     print("Tổng hai số m, n là:", m+n)  
7.     return  
8. Welcome()  
9. InputData()
```

Kết quả thực hiện chương trình trên như sau:

```
Xin chào !  
Nhập số m: 12  
Nhập số n: 35  
Tổng hai số m, n là: 47
```

2.2. Định nghĩa hàm số có giá trị

Định nghĩa hàm số có giá trị.

```
def <tên hàm số>(<tham số>):  
    <các lệnh ...>  
    return <giá trị>  
    <các lệnh ...>
```

Chú ý:

- Trong phần thân của hàm số bắt buộc phải có lệnh **return <giá trị>**, lệnh này có tác dụng dừng hàm số và trả lại giá trị cho hàm số.

Ví dụ sau thiết lập hàm số **Luythua(x,m,n)** dùng để tính hàm lũy thừa 2 bậc theo công thức sau:

$$\text{Luythua}(x,m,n) = x^{m^n}$$

```
>>> def Luythua(x,n,m):  
        return (x**n)**m
```

```
>>> Luythua(2,2,3)  
64
```

3. Kiểu truyền dữ liệu của tham số trong hàm số

Khi gọi hàm trong Python, dữ liệu được truyền thông qua tham số vào bên trong hàm được gọi là **đối số** của hàm.

Kiểu truyền đối số của hàm trong Python sẽ luôn là truyền giá trị, những gì xảy ra bên trong hàm số không ảnh hưởng đến bên ngoài. Bên trong mỗi hàm số, Python tạo một Namespace riêng để quản lý các biến nhớ bên trong hàm số, và Namespace này độc lập với Namespace tại nơi hàm số này được gọi.

Trong ví dụ đơn giản sau, hàm số **tinh()** được thiết lập có một tham số duy nhất. Biến nhớ tổng thể **n** được xác định bên ngoài hàm số = 5. Bên trong hàm số, giá trị **n** sẽ giảm đi 1 đơn vị (thành 4). Tuy nhiên bên ngoài hàm số, giá trị biến **n** không thay đổi.

```
n = 5  
>>> def tinh(n):  
        n = n-1  
        return n  
  
>>> tinh(n)  
4  
>>> n  
5  
>>>
```

Giá trị biến nhớ **n** không thay đổi khi thực hiện xong hàm số **tinh(n)**.



Trong Python, mọi hàm số đều truyền dữ liệu qua đối số theo kiểu truyền giá trị. Tất cả các biến mới được định nghĩa trong hàm số và các biến được truyền qua đối số đều chỉ có tác dụng bên trong hàm số.

4. Phạm vi áp dụng hàm số

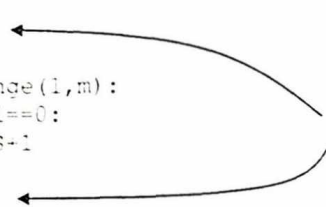
Hàm số trong Python có thể được xác định bên trong một hàm số khác. Nếu được định nghĩa bên trong một hàm số khác thì hàm được định nghĩa đó chỉ có tác dụng bên trong phạm vi mà hàm đó được khởi tạo.

Còn nếu hàm số được xác định trực tiếp không nằm bên trong một hàm số nào đó thì hàm số đó sẽ có phạm vi hoạt động tổng thể.

Xét ví dụ sau. Chương trình sau khai báo hàm `Tinhtoan()`, trong đó định nghĩa 2 hàm con bên trong là `Tinh_1()` và `Tinh_2()`.

Tinhtoan.py

```
def Tinhtoan(m):  
    def Tinh_1(m):  
        S=0  
        for i in range(1,m):  
            if m % i==0:  
                S = S+1  
        return S  
    def Tinh_2(m):  
        S=Tinh_1(m)  
        if S==1:  
            print(m,"là số nguyên tố")  
        else:  
            print(m,"là hợp số")  
        return  
    Tinh_2(m)  
    return
```



2 hàm số **Tinh_1()** và **Tinh_2()** được định nghĩa bên trong hàm **Tinhtoan()**.

Kết quả chạy chương trình này, hàm `Tinhtoan(m)` chạy như sau.

```
>>> Tinhtoan(10)  
10 là hợp số  
>>> Tinhtoan(17)  
17 là số nguyên tố
```

Tuy nhiên 2 hàm con bên trong không được xác định bên ngoài hàm `Tinhtoan()`. Kiểm tra trên cửa sổ lệnh của Python cho kết quả như sau.

```
>>> Tinh_2(10)  
Traceback (most recent call last):  
  File "<pyshell=68>", line 1, in <module>  
    Tinh_2(10)  
NameError: name 'Tinh_2' is not defined  
>>>
```

Chú ý quan trọng:

Trên thực tế lập trình Python, người ta ít sử dụng phương pháp định nghĩa hàm số lồng ghép nhiều mức như ví dụ trên.

5. Phạm vi của biến nhớ và tham số trong hàm số

Chúng ta sẽ tìm hiểu sâu hơn cách hàm số xử lý khi thực hiện các thao tác với biến nhớ hoặc tham số bên trong hàm số. Xét 2 ví dụ sau:

Ví dụ 1:

```
>>> def f(n):  
    n = n + 2  
    print(n)  
>>>
```

Ví dụ 2:

```
>>> m = 2  
>>> def g():  
    m = m + 2  
    print(m)  
>>>
```

Trong ví dụ 1, thực hiện lệnh, chẳng hạn lời gọi:

```
>>> f(10)
```

Giá trị 10 được truyền vào hàm số và biến nhớ n được định nghĩa bên trong hàm số, kết quả in ra giá trị $n + 2 = 12$. Biến nhớ n chỉ có ý nghĩa bên trong hàm số $f()$.

Ví dụ khác khi thực hiện hàm $f()$ của ví dụ 1.

```
>>> n = 15  
>>> f(n)  
17  
>>> n  
15  
>>>
```

Biến nhớ n được gán = 15 bên ngoài hàm $f()$. Khi chạy $f(n)$, một biến nhớ n khác được tạo bên trong hàm số và được gán giá trị 15. Biến nhớ n bên trong $f()$ không liên quan đến biến nhớ n bên ngoài $f()$.

Xét ví dụ 2.

```
>>> g()  
Traceback (most recent call last):  
  File "<pyshell#41>", line 1, in <module>  
    g()  
  File "<pyshell#37>", line 2, in g  
    m = m + 2  
UnboundLocalError: local variable 'm' referenced before  
assignment  
>>>
```

Chương trình báo lỗi vì:

- Mặc dù bên ngoài hàm số $g()$ đã xác định biến nhớ $m = 2$. Nhưng chương trình sẽ không hiểu lệnh $m = m + 2$ bên trong $g()$ vì lệnh $m = m + 2$ yêu cầu tạo một biến nhớ mới trong $g()$ có tên "m" nhưng lại gán giá trị biểu thức $m + 2$ nên bị lỗi.

- Muốn sửa lỗi này có thể sửa hàm $g()$ như sau:

```
>>> def g():  
    a = m + 2  
    print(a)  
>>> g()  
4  
>>>
```

hoặc đơn giản hơn như sau:

```
>>> def g():  
    print(m + 2)
```

Trong cả 2 trường hợp này, chương trình sẽ tìm giá trị của biến nhớ m bên trong hàm số $g()$, nếu không tìm thấy chương trình sẽ tìm giá trị bên ngoài $g()$ và tìm ra được $m = 2$.

6. Tham số thực sự và tham số lựa chọn

6.1. Tham số thực sự của hàm số

Chúng ta sẽ tìm hiểu sâu hơn về cách truyền dữ liệu qua các tham số của hàm số. Trong các ví dụ đã đưa ở phần trên, tất cả các tham số đều bắt buộc phải truyền tham số từ lời gọi hàm số, các tham số này gọi là **tham số thực sự (actual parameter)**. Ví dụ với hàm số sau cả 2 tham số x, y đều là các tham số thực sự.

```
>>> def f1(x, y):  
    return x*y + x - y
```

Hàm này có thể gọi đơn giản như sau:

```
>>> f1(2, 5)
```

```
7
```

```
>>>
```

Tuy nhiên không thể gọi hàm nếu thiếu tham số truyền vào, ví dụ:

```
>>> f1(5)
```

```
Traceback (most recent call last):
```

```
File "<pyshe11=84>", line 1, in <module>
```

```
    f1(5)
```

```
TypeError: f1() missing 1 required positional argument: 'y'
```

```
>>>
```

Chú ý thứ tự truyền tham số là quan trọng, ví dụ:

```
>>> f1(2, 5) ← Lời gọi này, hàm số sẽ hiểu x = 2, y = 5
```

```
7
```

```
>>> f1(5, 2) ← Còn lời gọi này thì x = 5, y = 2
```

```
13
```

```
>>>
```

Python cho phép gọi hàm số theo cách khác: thiết lập gán trực tiếp tên tham số trong lời gọi hàm. Ví dụ có thể gọi `f1(x = 2, y = 5)`. Cách gọi này có tên là gán tên biến trực tiếp.

Khi gọi hàm bằng cách gán tên trực tiếp thì có thể thay đổi thứ tự các tham số mà hàm vẫn hiểu được, ví dụ.

```
>>> f1(y=5, x=2)
```

```
7
```

```
>>>
```

→ Khi gọi hàm bằng cách viết, gán trực tiếp tên tham số thì thứ tự các tham số có thể thay đổi.

6.2. Tham số lựa chọn của hàm số

Trước khi tìm hiểu về cách tạo ra các tham số lựa chọn của hàm số trong Python, chúng ta cùng quan sát lại các ví dụ với hàm số `print()`.

Ví dụ sau thực hiện liên tiếp 2 lệnh `print()`.

```
>>> print("Hà Nội", "thủ đô"); print("Việt Nam")
```

```
Hà Nội thủ đô
```

```
Việt Nam
```

```
>>>
```

Chúng ta thấy gì?

- Lệnh tự động chèn các khoảng trống giữa các đối tượng cần in.

- Kết thúc lệnh sẽ tự động xuống dòng.

Quan sát lệnh tiếp theo, chúng ta sử dụng thêm 2 lựa chọn của lệnh là **sep** = <...> và **end** = <...>.

```
>>> print("Hà Nội","thủ đô",sep = "***",end= " ");print("Việt Nam")
Hà Nội***thủ đô  Việt Nam
>>>
```

- Tham số **sep** = <...> sẽ dùng giá trị này thay thế cho thể hiện ngăn cách giữa các đối tượng cần in ra của lệnh.

- Tham số **end** = <...> sẽ xác định thông tin được chèn vào cuối của lệnh **print()**.

- Mặc định, giá trị của tham số **sep** là dấu cách " ", của tham số **end** là ký tự điều khiển xuống dòng "\n".

Cú pháp đầy đủ hơn của **print()** là:

```
print(value1, value2, ..., valueN, [sep = <ký tự, dấu ngăn cách>],
      [end = <xâu kết thúc lệnh>])
```

Các tham số dạng **sep** = <...> hay **end** = <...> trong lệnh **print()** trên được gọi là các **tham số lựa chọn**. Các tham số lựa chọn có ý nghĩa sau:

3

0.000000

0.000000

0.000000

- Tham số lựa chọn luôn cần gán mặc định trong định nghĩa của hàm.

- Trong lời gọi hàm có thể vắng mặt tham số lựa chọn, khi đó giá trị mặc định của tham số này sẽ được gán trực tiếp.

- Có thể có một hoặc nhiều tham số lựa chọn.

6.3. Định nghĩa hàm số có tham số lựa chọn

Python cho phép khởi tạo dễ dàng các hàm số có tham số lựa chọn. Cách mô tả các tham số lựa chọn là có thêm dấu = và sau đó là giá trị mặc định được gán cho tham số này.

<tham số lựa chọn>=<giá trị mặc định>

Cú pháp định nghĩa hàm số đầy đủ trong Python.

```
def <hàm số>(<tham số thực sự>,<tham số lựa chọn>=<mặc định>):
    <các lệnh của hàm số>
```

Chú ý quan trọng:

Các tham số lựa chọn bắt buộc phải được mô tả sau các tham số thực sự.

Ví dụ sau cho thấy không thể đặt các tham số lựa chọn trước tham số thực sự.

```
>>> def f(x=2, y):
    return y**x
SyntaxError: non-default argument follows default argument
```

Chú ý:

- Các tham số lựa chọn trong Python gọi là **non-default argument**.

- Các tham số thực sự trong Python gọi là **default argument**.

Với một ví dụ đơn giản về hàm số có tham số lựa chọn.

```
def power(x, exp = 2):  
    return x**exp
```

Chúng ta có thể thực hiện các lời gọi đa dạng như sau.

```
>>> power(5)  
25  
>>> power(5, exp = 4)  
625  
>>> power(5, 4)  
625  
>>>
```

7. Hàm định nghĩa trực tiếp

Trong Python ngoài cách định nghĩa hàm số chính thống thông qua từ khóa `def`, còn có một cách định nghĩa nhanh hàm số khác nếu hàm số được tính bằng công thức. Cách tạo hàm số này còn được gọi là hàm vô danh hay hàm **lambda**.

Hàm **lambda** bắt buộc phải trả về giá trị bằng một biểu thức tính toán. Hàm có thể có một hoặc nhiều tham số.

Cú pháp định nghĩa hàm **lambda** như sau.

```
<tên hàm> = lambda <tham số>:<biểu thức tính>
```

Chú ý phần <tham số> có thể có nhiều tham biến.

Ví dụ hàm `d()` được định nghĩa trực tiếp bằng công thức $d(x, y) = x^2 + y^2$. Hàm này được định nghĩa trực tiếp như sau:

```
d = lambda x, y: x*x + y*y  
>>> d(1,2)  
5  
>>> d(10, 15)  
325  
>>> type(d)  
<class 'function'>
```

8. Một vài ví dụ hàm số

Ví dụ 1. Hàm **Nguyento(m)** đơn giản sau sẽ trả lại 1 nếu m là nguyên tố, ngược lại sẽ trả về 0 nếu m là hợp số.

```
1. def Nguyento(m):  
2.     S=0  
3.     for i in range(1,m):  
4.         if m % i == 0:  
5.             S = S+1  
6.     if S == 1:  
7.         return 1  
8.     else:  
9.         return 0
```

Chương trình sau mô tả một cách dùng đơn giản của hàm này. Chương trình yêu cầu nhập một số tự nhiên, sau đó thông báo ra màn hình số này là nguyên tố hay hợp số.

Nguyento.py

```

1. def Nguyento(m):
2.     S=0
3.     for i in range(1,m):
4.         if m % i == 0:
5.             S = S+1
6.         if S == 1:
7.             return 1
8.         else:
9.             return 0
10.
11. m = int(input("Nhập số tự nhiên m: "))
12. if Nguyento(m)==1:
13.     print(m,"là số nguyên tố")
14. else:
15.     print(m,"không là số nguyên tố")

```

Thuật toán: đếm số các ước số thực sự của m, lưu vào biến đếm S. Nếu S = 1 thì m là nguyên tố.

Ví dụ 2 đơn giản nhưng khá thú vị. Hàm số sẽ trả lại một dãy số (chứ không phải là một giá trị duy nhất).

Hàm Dayso(m) sau sẽ trả lại 2 giá trị là số các ước số thực sự của m và số các ước số nguyên tố của m.

Ham_dayso.py

```

1. def Dayso(m):
2.     S1=S2=0
3.     for i in range(1,m):
4.         if m % i==0:
5.             S1= S1+1
6.         for k in range(2,m):
7.             if m%k == 0:
8.                 s=0
9.                 for i in range(1,k):
10.                    if k%i == 0:
11.                        s=s+1
12.                    if s==1:
13.                        S2=S2+1
14.         return S1,S2
15.
16. m=int(input("Nhập số tự nhiên m: "))
17. print(Dayso(m))

```

S1 = số các ước số thực sự của m tính cả 1.

S2 = biến đếm số các ước số thực sự của m là nguyên tố.

Kết quả chạy chương trình này có thể như sau:

```

Nhập số tự nhiên m: 20
(5, 2)
Nhập số tự nhiên m: 100
(8, 2)

```



Câu hỏi - bài tập

1. Sau khi thực hiện đoạn chương trình sau thì giá trị kq là bao nhiêu?

```
1. kq = 1
```



```
2. def calc(x):
3.     kq = 4*x-1
4.     return kq
5. calc(7)
```

a) 27

b) 1

c) 7

d) 0

2. Đoạn chương trình sau sẽ in ra kết quả bao nhiêu:

```
1. def f(x):
2.     return x ** 2, x**x**x
3. x, y = f(2)
4. print(x+y)
```

a) 10

b) 20

c) 16

d) Chương trình báo lỗi.

3. Viết các hàm **lambda** mô tả các biểu thức sau:

a) $x + y + z$

b) $x ** (y + z)$

c) $(x + y) ** 2$

4. Viết hàm số $f(n)$ với tham số n là số tự nhiên theo yêu cầu sau.

a) Hàm trả lại số các ước số thực sự của n .

b) Hàm trả lại 1 nếu n là nguyên tố, ngược lại hàm trả về 0.

c) Hàm đếm số các ước số lẻ của n .

d) Hàm đếm số các ước số nguyên tố của n .

e) Hàm tính tổng tất cả các ước số thực sự của n .

5. Viết hàm số $f(m, n)$ với m, n là số tự nhiên theo yêu cầu sau.

a) Hàm đếm số các ước chung của các số m, n .

b) Hàm đếm số các ước số của m nhưng không là ước số của n .

c) Hàm tính tổng các ước số chung của m và n .

6. Viết hàm số Radian(d), tham số đầu vào là số thực là số đo góc theo độ bình thường. Hàm sẽ trả lại số đo của d theo radian. Công thức chuyển đổi là $R = d \times \pi / 180$.

7. Viết hàm số Degree(R), tham số đầu vào là số đo góc theo đơn vị radian. Hàm sẽ trả lại số đo góc theo độ bình thường. Công thức chuyển đổi là: $d = R \times 180/\pi$.

8. Sau đoạn chương trình sau, biến nhớ R nhận giá trị nào?

```
>>> def f(x, y, R):
```

```
R = x**2 + y**2
```

```
return
```

```
>>> R = 1
```

```
>>> f(3,4,R)
```

a) 25

b) 1

c) 0

d) Ngẫu nhiên.

8. Một hàm (thủ tục) khi khai báo có 3 tham số, ví dụ, $f(x, y, z)$. Nhưng khi gọi hàm, lời gọi chỉ có 2 tham số, ví dụ $f(x,y)$. Điều đó có thể thực hiện được hay không trong Python?

9. Hàm $f(x,y)$ được xác định như sau:

```
>>> def f(x, y):
```

```
    return x**3 + y**2
```

Khi đó lời gọi hàm $f(y=1,x=3)$ sẽ trả lại giá trị nào?

a) 10

b) 28

c) Chương trình báo lỗi.

10. Viết thủ tục **convert()** thực hiện các công việc sau:

- Yêu cầu nhập từ bàn phím số giờ, phút, giây (tính theo số nguyên).
- Sau đó đổi tổng số thời gian này ra ngày (tính theo float).
- Hiển thị kết quả ra màn hình.

Ví dụ thủ tục này chạy như sau.

```
Nhập số giờ: 100
```

```
Nhập số phút: 43
```

```
Nhập số giây: 56
```

```
4.197175925925926
```

11. Viết hàm số **your_weight(w, m)** dùng để tính trọng lượng thực tế của bạn trên một hành tinh nào đó. Hàm này có 2 tham số:

- w: Trọng lượng của bạn trên Trái đất, tính theo N (Newton).
- m: Trọng lực (Gravity) của hành tinh mà bạn cần tính, giá trị này tính theo đơn vị mét/giây² (m/s²).

Chú ý rằng gravity của Trái đất là 9.8m/s².

Công thức tính trọng lượng của bạn trên hành tinh có gravity = m được tính như sau:

Cân nặng của bạn sẽ là $w*m/9.8$.

Ví dụ hàm số này chạy như sau:

```
>>> your_weight(70.3,9.8) # trọng lượng trên Trái đất
70.3
```

```
>>> your_weight(70.3,23.1) # trọng lượng trên sao Mộc.
165.70714285714286
```

12. Chúng ta đã biết đơn vị đo nhiệt độ hiện nay trên thế giới có 3 loại:

- Nhiệt độ tính theo Celsius (nhiệt độ C).
- Nhiệt độ tính theo Kelvin (nhiệt độ K).
- Nhiệt độ tính theo Fahrenheit (nhiệt độ F).

Các công thức sau cho biết quan hệ toán học giữa các nhiệt độ trên.

$$T_c = \frac{5}{9}(T_f - 32)$$

$$T_k = T_c + 273.15$$

Viết thủ tục **convertCK()** thực hiện các công việc sau:

- Yêu cầu nhập từ màn hình giá trị nhiệt độ T_c tính theo C.
- Tính toán và đưa ra màn hình nhiệt độ T_k tính theo K.

13. Tương tự viết các thủ tục chuyển đổi nhiệt độ **convertKC()**, **convertFK()**, **convertKF()**, **convertFC()** và **convertCF()**

14. Đoạn chương trình sau có lỗi gọi hàm $f(x)$ ở cuối.

```
>>> def f(x):
    return x+y+z
>>> f(1)
```

Kết quả thực hiện trả lại giá trị nào?

- 1
- 0
- Chương trình báo lỗi.

15. Viết các thủ tục thực hiện các công việc sau:

- Tham số là số tự nhiên n , in ra màn hình kết quả tổng các số $1 + 2 + \dots + n$.
- Tham số là 2 số tự nhiên m, n , in ra thông báo: "Hai số này là nguyên tố cùng nhau" hoặc "Hai số này không là nguyên tố cùng nhau".
- Tham số là số tự nhiên s chỉ số giây, thông báo số giờ, phút, giây mà giá trị s này chuyển sang.
- Tham số là hai số thực a, b , trả lại giá trị là trị tuyệt đối $|a - b|$.

16. Viết thủ tục **NhapDL()** có tính năng yêu cầu người dùng nhập số tự nhiên từ bàn phím. Chương trình sẽ yêu cầu nhập liên tục cho đến khi số cần nhập > 50 thì dừng lại.

17. Viết hàm số $f(m, n)$ với các tham số là các số tự nhiên. Hàm sẽ trả lại số nguyên tố lớn nhất đồng thời là ước chung của m và n . Nếu số đó không tồn tại thì trả lại 0.

18. Viết hàm số **factorial(n)** tính giá trị là tích các số $1.2....n$ (gọi là n giai thừa).

19. Viết hàm số **nhuan(year)** trả lại **True** nếu năm $year$ là nhuận và **False** nếu $year$ không là năm nhuận.

20. Viết hàm số (hàm dấu) **sign(x)** trả lại 1 nếu $x > 0$, trả lại 0 nếu $x = 0$ và trả lại -1 nếu $x < 0$.

Chương 5. Đối tượng

Mục tiêu:

- Khái niệm Đối tượng (Objects) và Lớp (Class) trong Python.
- Cách tạo và làm việc với lớp và các đối tượng.

Nội dung chính

1. Tổng quan về lập trình hướng đối tượng (OOP)

Ý tưởng chung của lập trình hướng đối tượng xuất phát từ mô hình thực tế xung quanh chúng ta: Tất cả những đồ vật, vật dụng hàng ngày, xung quanh chúng ta đều nằm trong các nhóm riêng của mình. Ví dụ: cái bàn, ghế nằm trong nhóm đồ dùng gia đình; máy tính, điện thoại, máy in nằm trong nhóm các thiết bị văn phòng. Trong xã hội con người cũng vậy, mỗi con người đều nằm trong các nhóm, ví dụ dân tộc, quốc tịch. Điều này cũng được mang vào các ngôn ngữ lập trình. Nhưng trong ngôn ngữ lập trình các tên gọi sẽ được thay đổi:

Từ **đồ vật** sẽ được thay đổi trong lập trình thành **Đối tượng** (Objects).

Từ **nhóm** sẽ được thay đổi trong lập trình thành **Lớp** (Class).

Như vậy có thể nói trong các ngôn ngữ lập trình, tất cả mọi thứ chúng ta biết và làm quen đều thuộc và liên quan đến khái niệm **Lớp - Đối tượng**.

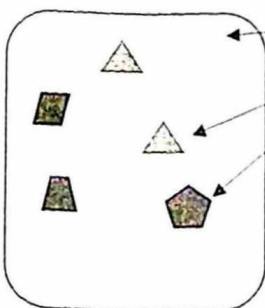
Ví dụ, chúng ta đã biết về các kiểu dữ liệu như xâu ký tự (str), kiểu số nguyên (int) hay kiểu số thực (float). Khi một biến nhớ được khai báo, ví dụ x, thì x là *đối tượng*, kiểu dữ liệu của x sẽ là *lớp*.

```
>>> x = "Việt Nam"
>>> type(x)
<class 'str'>
>>> x = 12
>>> type(x)
<class 'int'>
>>> x = 12.3
>>> type(x)
<class 'float'>
```

Đối tượng x được gán dữ liệu và tự động thuộc về các lớp **str**, **int** và **float**.

Các lớp **str**, **int**, **float** được thiết lập sẵn, mặc định trong Python.

Chúng ta cùng quan sát sâu hơn nữa vào mô hình đối tượng - nhóm trên thực tế. Ví dụ nhóm các đồ dùng văn phòng, đối tượng là bàn, ghế làm việc.



Nhóm các đồ dùng văn phòng.

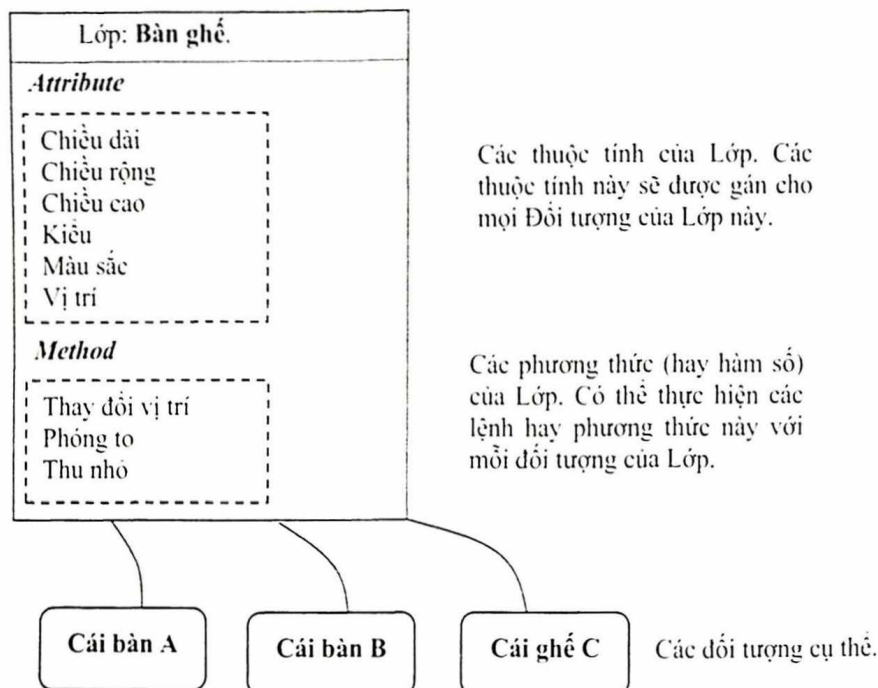
Đối tượng: Bàn, ghế làm việc.

Chúng ta thấy mỗi đối tượng trong nhóm đều có những đặc thù, tính chất riêng biệt, ví dụ màu sắc, kích thước, kiểu dáng, giá bán, ...

Những tính chất này gọi là **thuộc tính (attribute)** của các đối tượng. Mỗi đối tượng có các thuộc tính riêng.

Tương tự, khái niệm Lớp (Class) và Đối tượng (Objects) trong các ngôn ngữ lập trình cũng như vậy. Có một điểm khác biệt: Các tính chất, thuộc tính của đối tượng trong Lớp cần được định nghĩa trước khi khởi tạo Lớp. Bên cạnh khái niệm thuộc tính (Attribute), đối tượng còn có thêm khái niệm hành vi, còn gọi là phương thức (Method) hay chính là các hàm, thủ tục của đối tượng. Các phương thức hay hàm, thủ tục này cũng được định nghĩa trước trong các Lớp.

Như vậy mô hình Đối tượng - Lớp trong ngôn ngữ lập trình bậc cao được mô tả trong sơ đồ đơn giản sau.



Các Đối tượng được khởi tạo từ Lớp tương ứng. Mỗi đối tượng có các thuộc tính riêng của mình. Chúng ta gọi mỗi đối tượng là một **thể hiện riêng** (Instance) của Lớp đã cho.

Cách lập trình làm việc với các Lớp và Đối tượng được gọi là lập trình hướng đối tượng (**Object Oriented Programming - OOP**). Python là ngôn ngữ lập trình hướng đối tượng hoàn toàn.

2. Khởi tạo Lớp và Đối tượng

Trong mục này chúng ta sẽ làm quen với các lệnh khởi tạo và làm việc với các Lớp và Đối tượng trong Python.

Lệnh tạo Lớp (Class) trong Python đơn giản như sau.

```
class <Tên Lớp>:  
    <phần mô tả Lớp>
```

Trong đó <phần mô tả Lớp> là các khai báo thuộc tính và phương thức cho lớp này.

Xét một vài ví dụ cụ thể sau.

Ví dụ 1: Tạo một Lớp có tên **Hocsinh** với các thuộc tính như name (tên), age (tuổi), male (nam hay nữ) và school (tên trường học của học sinh này).

```
>>> class Hocsinh:
    name = "Bùi Vũ Huy"
    age = 14
    gender = 1
    school = "THCS Trung Vương"
```

Mô tả các thuộc tính của lớp.

Trong ví dụ trên một Lớp có tên **Hocsinh** đã được khai báo. Lớp này có 4 thuộc tính name, age, gender, school. Các thuộc tính này được tạo và gán giá trị mặc định trong phần thân của lớp.

Trong ví dụ sau, chúng ta sẽ khai báo một đối tượng mới **sh1** của lớp **Hocsinh**.

```
>>> sh1 = Hocsinh()
>>> sh1.name
'Bùi Vũ Huy'
>>>
```

Để truy cập thuộc tính cụ thể của đối tượng chúng ta dùng cách viết sau.

<Đối tượng>.<thuộc tính>

Trong ví dụ tiếp theo, chúng ta tạo một đối tượng khác, sh2, của lớp Hocsinh. Đối tượng này cũng được gán thuộc tính mặc định như sh1. Muốn thay đổi thuộc tính chúng ta phải dùng lệnh gán trực tiếp.

Có thể dễ dàng thay đổi thuộc tính của đối tượng bằng lệnh gán trực tiếp như sau:

<Đối tượng>.<thuộc tính> = <giá trị mới>

Đoạn các lệnh tương tác với sh2 như sau.

```
>>> sh2 = Hocsinh()
>>> sh2.name = "Nguyễn Thị Hoa"
>>> sh2.name
'Nguyễn Thị Hoa'
>>>
```

Bây giờ chúng ta bổ sung khai báo phương thức cho Lớp Hocsinh này.

```
>>> class Hocsinh:
    name = "Bùi Vũ Huy"
    age = 14
    male = 1
    school = "THCS Trung Vương"
    def show(self):
        print("Name = ", self.name)
        print("Age = ", self.age)
        print("School= ", self.school)
        print("Gender = ", self.male)
    def update(self, name, age, male, school):
        self.name = name
        self.age = age
        self.male = male
        self.school = school
```

self là từ dùng để chỉ bản thân đối tượng, rất hay dùng trong khai báo phương thức. Có thể dùng tên khác nhưng Python khuyên nên sử dụng tên self mặc định này,

Chú ý: Phương thức **update()** có chức năng thay đổi thuộc tính của đối tượng. Trong các tham số của phương thức này bắt buộc phải có tham số đóng vai trò là chủ thể đối tượng cần thay đổi. Có thể dùng tên tham số là **self** hay bất kỳ một tên nào khác.

Bây giờ chúng ta có thể thực hiện trực tiếp các phương thức của đối tượng. Cách thực hiện phương thức trực tiếp như sau:

<Đối tượng>.<phương thức>

Ví dụ:

```
>>> hsl=Hocsinh()
>>> hsl.show()
Name = Bùi Vũ Huy
Age = 14
School= THCS Trung Vương
Gender = 1
>>> hsl.update("Hà", 15, 0, "THCS Chu Văn An")
>>> hsl.school
'THCS Chu Văn An'
>>>
```

Đối tượng **hsl** được gán thuộc tính mặc định.

Thay đổi các thuộc tính cho đối tượng này.

Ví dụ 2: Tạo một Lớp có tên BanGhe.

```
class BanGhe:
    height = 60
    width = 100
    length = 180
    color = "Red"
    X = 0
    Y = 0
    def change(self,X,Y):
        self.X = X
        self.Y = Y
    def changecolor(self,color):
        self.color = color
    def changedim(self,height,width,length):
        self.height = height
        self.width = width
        self.length = length
```

Các thuộc tính của lớp **BanGhe**: chiều cao, chiều rộng, chiều dài, màu sắc, tọa độ X, Y trên mặt phẳng.

3 phương thức của lớp này: thay đổi vị trí; thay đổi màu sắc và thay đổi kích thước.

3. Lệnh **dir()** và **__init__()**

Khi tạo một lớp mới tức là chúng ta đã tạo ra một kiểu dữ liệu mới. Các kiểu dữ liệu này được gọi là kiểu dữ liệu do người dùng khởi tạo.

Xét ví dụ lớp Hocsinh đã định nghĩa trong phần trên, **hsl** là một đối tượng của lớp này.

```
>>> type(Hocsinh)
<class 'type'>
>>> hsl=Hocsinh()
>>> type(hsl)
<class '__main__.Hocsinh'>
```

Hocsinh sẽ thuộc kiểu type, tức là kiểu dữ liệu mới.

hsl thuộc kiểu Hocsinh.

3.1. Lệnh dir()

Lệnh `dir` có cú pháp `dir(<đối tượng hoặc lớp>)` sẽ làm hiện tất cả các thuộc tính và phương thức của đối tượng / lớp tương ứng. Đây là một lệnh rất hữu ích.

Ví dụ kết quả của lệnh `dir()` với lớp `Hocsinh` như sau.

```
>>> dir(Hocsinh)
['_class_', '_delattr_', '_dict_', '_dir_', '_doc_',
'_eq_', '_format_', '_ge_', '_getattr_', '_gt_',
'_hash_', '_init_', '_init_subclass_', '_le_', '_lt_',
'_module_', '_ne_', '_new_', '_reduce_', '_reduce_ex_',
'_repr_', '_setattr_', '_sizeof_', '_str_',
'_subclasshook_', '_weakref_', '_age_', '_gender_', '_name_',
'_school_', '_show_', '_update_']
>>>
```

Tất cả các phương thức bắt đầu và kết thúc bởi "_" đều là hệ thống và có tên cố định không thể thay đổi. Các thuộc tính và phương thức do chúng ta tạo ra sẽ được liệt kê ở cuối danh sách² (được in đậm).

Trong Python, các kiểu dữ liệu chính như `str`, `int` hay `float` đều là các lớp hệ thống có sẵn. Chúng ta cùng quan sát cấu trúc của các lớp hệ thống này.

Cấu trúc của lớp `str`.

```
>>> dir(str)
['_add_', '_class_', '_contains_', '_delattr_',
'_dir_', '_doc_', '_eq_', '_format_', '_ge_',
'_getattr_', '_getitem_', '_getnewargs_', '_gt_',
'_hash_', '_init_', '_init_subclass_', '_iter_',
'_le_', '_len_', '_lt_', '_mod_', '_mul_', '_ne_',
'_new_', '_reduce_', '_reduce_ex_', '_repr_', '_rmod_',
'_rmul_', '_setattr_', '_sizeof_', '_str_',
'_subclasshook_', '_capitalize_', '_casefold_', '_center_', '_count_',
'_encode_', '_endswith_', '_expandtabs_', '_find_', '_format_',
'_format_map_', '_index_', '_isalnum_', '_isalpha_', '_isascii_',
'_isdecimal_', '_isdigit_', '_isidentifier_', '_islower_', '_isnumeric_',
'_isprintable_', '_isspace_', '_istitle_', '_isupper_', '_join_', '_ljust_',
'_lower_', '_lstrip_', '_maketrans_', '_partition_', '_replace_', '_rfind_',
'_rindex_', '_rjust_', '_rpartition_', '_rsplit_', '_rstrip_', '_split_',
'_splitlines_', '_startswith_', '_strip_', '_swapcase_', '_title_',
'_translate_', '_upper_', '_zfill_']
>>>
```

Lệnh `dir()` có thể áp dụng cho một đối tượng cụ thể. Python tự động xác nhận lớp của đối tượng và hiển thị các thuộc tính và phương thức của lớp tương ứng.

Muốn tra cứu cụ thể các hàm, phương thức của lớp (hay đối tượng) chúng ta dùng lệnh `help()` như sau:

`help(<đối tượng>.<phương thức>)`

Ví dụ khi gán `s = "Hà Nội"`, Python tự động hiểu `s` là đối tượng của lớp `str`.

² Chúng ta sẽ học trong bài sau danh sách kết quả của lệnh `dir()` là một dữ liệu kiểu List. Kiểu List trong Python sẽ được gọi với các tên khác như Mảng hay Danh sách.

Các lệnh sau sẽ tra cứu phương thức upper() đối với kiểu dữ liệu str và thực hiện phương thức này. Hàm upper() sẽ biến đổi chuỗi ký tự hiện thời lên toàn các ký tự in hoa.

```
>>> s = "Hà Nội"
>>> help(s.upper)
Help on built-in function upper:
upper() method of builtins.str instance
    Return a copy of the string converted to uppercase.
>>> s.upper()
'HÀ NỘI'
```

Một hàm khác, hàm split() sẽ tách các từ đơn của chuỗi ký tự hiện thời, ví dụ với chuỗi "Hà Nội" kết quả sẽ là:

```
>>> s.split()
['Hà', 'Nội']
```

Các phương thức của lớp dữ liệu kiểu int (số nguyên).

```
>>> dir(int)
['_abs_', '_add_', '_and_', '_bool_', '_ceil_',
'_class_', '_delattr_', '_dir_', '_divmod_', '_doc_',
'_eq_', '_float_', '_floor_', '_floordiv_', '_format_',
'_ge_', '_getattr_', '_getattribute_', '_getnewargs_', '_gt_',
'_hash_', '_index_', '_init_', '_init_subclass_',
'_int_', '_invert_', '_le_', '_lshift_', '_lt_',
'_mod_', '_mul_', '_ne_', '_neg_', '_new_', '_or_',
'_pos_', '_pow_', '_radd_', '_rand_', '_rdivmod_',
'_reduce_', '_reduce_ex_', '_repr_', '_rfloordiv_',
'_rlshift_', '_rmod_', '_rmul_', '_ror_', '_round_',
'_rpow_', '_rrshift_', '_rshift_', '_rsub_',
'_rtruediv_', '_rxor_', '_setattr_', '_sizeof_',
'_str_', '_sub_', '_subclasshook_', '_truediv_',
'_trunc_', '_xor_', 'bit_length', 'conjugate', 'denominator',
'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

Còn đây là thông tin của kiểu dữ liệu float (số thực).

```
>>> dir(float)
['_abs_', '_add_', '_bool_', '_class_', '_delattr_',
'_dir_', '_divmod_', '_doc_', '_eq_', '_float_',
'_floordiv_', '_format_', '_ge_', '_getattr_',
'_getattribute_', '_getformat_', '_getnewargs_', '_gt_',
'_hash_', '_init_', '_init_subclass_', '_int_', '_le_', '_lt_',
'_mod_', '_mul_', '_ne_', '_neg_', '_new_', '_pos_',
'_pow_', '_radd_', '_rdivmod_', '_reduce_',
'_reduce_ex_', '_repr_', '_rfloordiv_', '_rmod_',
'_rmul_', '_round_', '_rpow_', '_rsub_', '_rtruediv_',
'_set_format_', '_setattr_', '_sizeof_', '_str_',
'_sub_', '_subclasshook_', '_truediv_', '_trunc_',
'as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'imag',
'is_integer', 'real']
```


3.2. Lệnh `__init__()`

Trong các phương thức hệ thống của lớp, thủ tục `__init__()` đóng vai trò rất quan trọng. Phương thức này có chức năng gán tự động thuộc tính cho đối tượng khi mới khởi tạo. Mặt khác khi đã có `__init__()` thì không cần khai báo thuộc tính bên trong phần mô tả của lớp nữa.

Cú pháp của phương thức này được mô tả trong ví dụ sau, chúng ta định nghĩa lại lớp `Hocsinh`.

```
class Hocsinh:
    def __init__(self, name, age, gender, school):
        self.name = name
        self.age = age
        self.gender = gender
        self.school = school
    def show(self):
        print("Name = ", self.name)
        print("Age = ", self.age)
        print("School= ", self.school)
        print("Gender = ", self.gender)
```

Chú ý: phương thức `__init__()` cần có tham số (`self`) đóng vai trò đối tượng được gán các thuộc tính ban đầu.

Nếu `__init__()` đã được định nghĩa thì lệnh khởi tạo một đối tượng sẽ được thực hiện như sau với cú pháp lấy từ mô tả của phương thức này.

Khi khởi tạo đối tượng, phương thức `__init__()` sẽ được thực hiện ngay cho đối tượng này để gán các thuộc tính ban đầu.

```
hsl = Hocsinh("Hà", 14, 1, "THCS Trung Vương")
```

Các tham số của lệnh khởi tạo đối tượng giống như `__init__()`.

Kết quả của lệnh khởi tạo đối tượng mới như sau.

```
>>> hsl.show()
Name = Hà
Age = 14
School= THCS Trung Vương
Gender = 1
```

4. Các toán tử chồng (Overloaded Operator)

Khái niệm **toán tử chồng** (overloaded operator) có trong tất cả các ngôn ngữ lập trình hướng đối tượng.

Toán tử chồng ở đây được hiểu là các toán tử, ví dụ phép toán `+`, `-`, `x` nhưng khi áp dụng trong những hoàn cảnh khác nhau thì sẽ được hiểu theo các nghĩa khác nhau. Hay nói cách khác các toán tử chồng là những toán tử mà ý nghĩa thực tế của chúng phụ thuộc vào hoàn cảnh hiện thực (context) khi thực hiện toán tử này.

Điều đặc biệt trên được giải thích như sau: Các phép tính hay toán tử trong ngôn ngữ lập trình hướng đối tượng OOP thực chất là các phương thức, hay hàm số được định nghĩa trong các lớp. Khi thực hiện một phép tính, phương thức tương ứng sẽ được gọi phụ thuộc

vào kiểu của các tham số tham gia vào toán tử đó. Vì với mỗi lớp, kiểu dữ liệu khác nhau thì các toán tử, phép tính đó được định nghĩa khác nhau.

Chúng ta sẽ liệt kê ở đây một số trường hợp của các toán tử chồng trong Python³.

Phép cộng +

Stt	Phép tính thực tế	Ý nghĩa
1	+ <giá trị số, int hoặc float>	Thiết lập dấu + (số dương) của giá trị đó.
2	+ <xâu ký tự, str>	Không có nghĩa.
3	<số int> + <số int>	Phép cộng 2 số nguyên, kết quả nguyên.
4	<số int> + <số float>	Phép cộng 2 số thực, kết quả số thực.
5	<số float> + <số float>	Phép cộng 2 số thực, kết quả số thực.
6	<số int, float> + <xâu ký tự, str>	Không có nghĩa.
7	<xâu ký tự> + <xâu ký tự>	Phép nối 2 xâu ký tự, kết quả là xâu ký tự được ghép.

Phép nhân *

Stt	Phép tính thực tế	Ý nghĩa
1	* <giá trị số hoặc xâu ký tự>	Không có nghĩa.
2	<số int> * <số int>	Phép nhân 2 số nguyên, kết quả nguyên.
3	<số int> * <số float>	Phép nhân 2 số thực, kết quả là số thực.
4	<số float> * <số float>	Phép nhân 2 số thực, kết quả là số thực.
5	<số int> * <xâu ký tự, str>	Ghép nối các xâu ký tự với số nguyên lần. Nếu <số int> < 0 thì kết quả là xâu rỗng.
6	<số float> * <xâu ký tự, str>	Không có nghĩa.
7	<xâu ký tự> * <xâu ký tự>	Không có nghĩa.

Ví dụ:

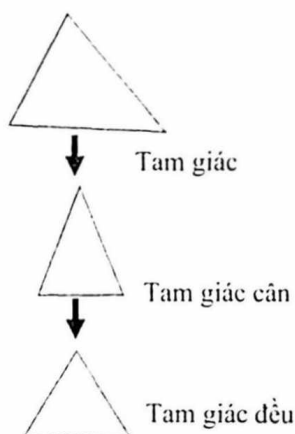
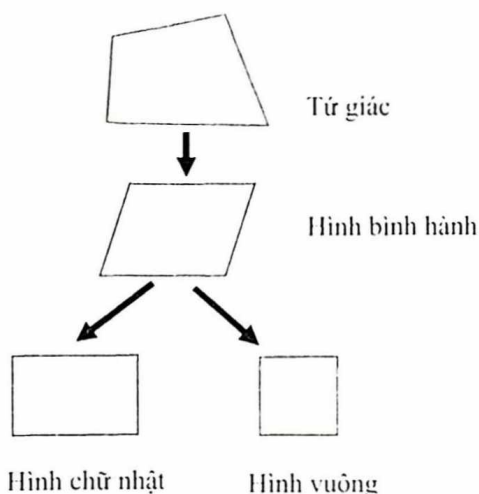
```
>>> s, n, x = "Hà Nội ", 12, 3.14
>>> 4*s           # ghép 4 lần xâu s
'Hà Nội Hà Nội Hà Nội Hà Nội '
>>> s + "Việt Nam" # ghép 2 xâu.
'Hà Nội Việt Nam'
>>> x*s           # không thể thực hiện phép toán này.
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    x*s
TypeError: can't multiply sequence by non-int of type 'float'
>>>
```

³ Trong chương này chỉ xét các kiểu dữ liệu cơ bản như int, float, str, chưa xét đến các dạng dữ liệu khác như danh sách, dãy, từ điển, tập hợp, ...

5. Lớp con và tính kế thừa

Trên thực tế, các lớp đối tượng có thể được kế thừa nhau nếu chúng được sinh ra dựa trên các đối tượng khác có tính chất phổ quát hơn.

Các hình sau mô tả các lớp đối tượng có tính kế thừa nhau.



Các hình chữ nhật, hình vuông là trường hợp riêng của hình bình hành, hình bình hành là trường hợp riêng của hình tứ giác.

Hình tam giác đều là trường hợp riêng của tam giác cân, hình tam giác cân là trường hợp riêng của hình tam giác.

Khi đã có một lớp đối tượng (lớp cha), chúng ta có thể tạo ra các lớp con của lớp đã cho. Cách tạo một lớp con như sau.

```
class <tên Lớp con>(<Lớp cha>):  
    <lệnh mô tả 1>  
    .....  
    <lệnh mô tả N>
```

Các đối tượng của lớp con sẽ kế thừa các thuộc tính, phương thức của lớp cha mà phù hợp với các thuộc tính của lớp con.

Xét ví dụ hình tứ giác - hình bình hành - hình chữ nhật, hình vuông ở trên.

Lớp `Tu_giac` được xây dựng với các thuộc tính `name`, độ dài 4 cạnh `a`, `b`, `c`, `d` và phương thức tính chu vi.

```
class Tu_giac:  
    def __init__(self, Name, a, b, c, d):  
        self.name = Name  
        self.a = a  
        self.b = b  
        self.c = c  
        self.d = d  
    def Chuvi(self):  
        return self.a + self.b + self.c + self.d
```

Lớp đối tượng hình bình hành được xây dựng là lớp con của lớp tứ giác. Đối tượng này có các thuộc tính là name, độ dài 2 cạnh a, b và góc alpha giữa chúng.

```
class Binh_hanh(Tu_giac):
    def __init__(self, Name, a, b, alpha = 45):
        self.name = Name
        self.a = a
        self.b = b
        self.alpha = alpha
    def Chuvi(self):
        return 2 * (self.a + self.b)
    def DT(self):
        import math as m = import module math4
        return self.a * self.b * m.sin(m.radians(self.alpha))
```

Chú ý: chúng ta viết lại 2 phương thức tính chu vi và diện tích của hình bình hành.

Sau đây là mô tả 2 lớp đối tượng hình chữ nhật và hình vuông, đều là lớp con của hình bình hành. Hình chữ nhật sẽ kế thừa phương thức tính chu vi của hình bình hành, còn với hình vuông cần viết lại cả 2 phương thức này.

```
class Chu_nhat(Binh_hanh):
    def __init__(self, Name, a, b):
        self.name = Name
        self.a = a
        self.b = b
    def DT(self):
        return self.a * self.b
class Hinh_vuong(Binh_hanh):
    def __init__(self, Name, a):
        self.name = Name
        self.a = a
        self.b = a
    def Chuvi(self):
        return 4 * self.a
    def DT(self):
        return self.a * self.a
```

Ví dụ đoạn chương trình sau.

```
h1 = Chu_nhat("h1", 5, 6)
h2 = Hinh_vuong("h2", 5.54)
print(h1.name)
print(h1.Chuvi())
print(h2.name)
print(h2.DT())
```

Khi thực hiện sẽ đưa ra kết quả như sau.

```
h1
22
h2
30.6916
```

⁴ Về khái niệm module và lệnh import xem chương 8.



Câu hỏi - bài tập

1. Đoạn chương trình sau khai báo một lớp và tạo một đối tượng cụ thể.

```
>>> class Hocsinh:  
    Ten = "Hà"  
>>> h = Hocsinh('Bình')
```

Hỏi kết quả đối tượng h được tạo ra có tên gì:

- a) Có tên "Hà".
- b) Có tên "Bình".
- c) Không thể khởi tạo được.

2. Đoạn chương trình sau khai báo lớp và tạo đối tượng cụ thể.

```
>>> class Hocsinh:  
    Ten = "Hà"  
>>> h = Hocsinh()  
>>> h.Ten = "Bình"
```

Hỏi kết quả đối tượng h được tạo ra có tên gì:

- a) Có tên "Hà".
- b) Có tên "Bình".
- c) Không thể khởi tạo được.

3. Thiết lập lớp **QuadrilSquare** bao gồm đối tượng là các hình chữ nhật hoặc hình vuông.

- Mỗi đối tượng sẽ có 2 thuộc tính là chiều dài **w** và chiều cao **h**.
- Các method chính của lớp này là các hàm tính chu vi (**Chuvi()**) và diện tích (**Dientich()**) của mỗi đối tượng.

4. Thiết lập lớp **Hocsinh** dùng để quản lý các đối tượng học sinh trong lớp em.

Các thuộc tính của mỗi học sinh bao gồm:

- Họ và tên.
- Địa chỉ.
- Chiều cao.
- Cân nặng.
- Học lực.

Lớp Hocsinh được thiết lập với các phương thức sau:

- Hàm **__init__()** thiết lập tên học sinh, địa chỉ, chiều cao và cân nặng.
- Hàm cập nhật địa chỉ mới.
- Hàm cập nhật chiều cao, cân nặng mới.
- Hàm cập nhật học lực học sinh.

5. Thiết lập lớp **Triple** bao gồm các đối tượng là một bộ 3 số thực (a, b, c). Thuộc tính của mỗi đối tượng bao gồm:

- Tên của đối tượng.

- Bộ 3 số a, b, c.
- Tổng 3 số này, ký hiệu S.

Thiết lập các phương thức sau với các đối tượng này:

- Hàm `__init__()` dùng để khởi tạo 1 bộ triple.
- Hàm `Update()` dùng để cập nhật giá trị S.
- Hàm `Change()` có chức năng thay đổi các giá trị a, b, c và đồng thời cập nhật S.
- Hàm `isTriangle()` kiểm tra xem bộ 3 số này có tạo thành một tam giác hay không.

6. Thiết lập lớp **Tam_giac** là lớp con của lớp **Triple** trong bài tập 5. Bổ sung các phương thức sau cho lớp con này:

- Hàm tính diện tích tam giác.
- Hàm tính bán kính vòng tròn ngoại tiếp tam giác.
- Hàm tính bán kính vòng tròn nội tiếp tam giác.

Lớp con **Tam_giac** này sẽ kế thừa được phương thức nào từ lớp cha **Triple**?

7. Có thể coi mỗi lớp là một kiểu dữ liệu mới được không? Giải thích lý do.

8. Trong ví dụ lớp Tứ giác từ mục 5, giả sử chúng ta bổ sung thêm phương thức thay đổi tên của đối tượng tứ giác này. Hỏi khi đó các đối tượng của các lớp con hình bình hành, hình chữ nhật, hình vuông có kế thừa được phương thức này hay không?

9. Thiết kế 3 lớp kế thừa nhau là tam giác, tam giác cân và tam giác đều. Trong đó lớp tam giác cân là con của lớp tam giác, lớp tam giác đều là lớp con của lớp tam giác cân. Thiết lập các phương thức tính chu vi, diện tích cho các đối tượng này.

10. Lớp X được định nghĩa như sau:

```
class X(object):
    def id(self):
        print("I'm an X object")
```

Khi đó nếu viết các lệnh:

```
x = X()
x.id()
```

thì kết quả như thế nào?

- "I'm an X object"
- I'm an X object
- Không thông báo gì.

11. Khi nào thì một thuộc tính của Lớp cha sẽ được kế thừa xuống cho Lớp con?

12. Khi nào thì một phương thức của Lớp cha sẽ được kế thừa xuống cho Lớp con?

13. Thiết lập lớp **Circle** (vòng tròn) bao gồm thuộc tính R (bán kính) và tọa độ tâm X, Y. Thiết lập các phương thức tính chu vi và diện tích của đối tượng hình tròn.

Chương 6. Kiểu dữ liệu List. Mảng một chiều

Mục tiêu:

- Làm quen với các kiểu dữ liệu mảng như List, Tuple và Range.
- Thao tác ban đầu với các kiểu dữ liệu trên.



Nội dung chính

Trong các bài học trên chúng ta đã làm quen với các kiểu dữ liệu cơ bản, đơn giản như số nguyên (int), số thực hay số dấu phẩy động (float), xâu ký tự (str), logic (bool). Đây là các kiểu dữ liệu đơn giản nhất và không có cấu trúc.

Khi cần tổ chức dữ liệu ở mức độ phức tạp hơn, chúng ta sẽ nghĩ đến khái niệm “cấu trúc dữ liệu” hay dữ liệu có cấu trúc. Có rất nhiều cấu trúc dữ liệu như ngăn xếp (stack), hàng đợi (queue), cây (tree), ... nhưng cấu trúc dữ liệu hay dùng nhất chắc chắn là dữ liệu mảng hay danh sách (array). Trong bài học này và bài tiếp theo chúng ta sẽ cùng tìm hiểu cấu trúc dữ liệu này.

List (danh sách) là một trong các kiểu dữ liệu quan trọng nhất của Python. Khác với hầu hết các ngôn ngữ lập trình bậc cao khác, **list** trong Python có rất nhiều tính chất đặc biệt hay và thú vị. Song hành với list, trong Python còn có 2 kiểu dữ liệu “tương tự” khác là **tuple** (dãy) và **range** (vùng các số nguyên)⁵.

Chúng ta sẽ cùng nhau tìm hiểu về kiểu dữ liệu này.

1. Khởi tạo List

List được hiểu là một dãy các giá trị được viết trong dấu ngoặc vuông, việc khởi tạo một list hay danh sách rất đơn giản, ví dụ như sau:

```
>>> a = [10, 4, 9, 8]
>>> type(a)
<class 'list'>
```

—————> **list** là kiểu dữ liệu quan trọng trong Python.

```
>>> a
[10, 4, 9, 8]
>>>
```

Như vậy danh sách (list) là một dãy các giá trị số được viết cách nhau bởi dấu phẩy và bao hàm trong dấu ngoặc vuông.

List được định nghĩa bởi [**giá trị 1**], [**giá trị 2**], ..., [**giá trị n**].

Giống như các kiểu dữ liệu mảng trong các ngôn ngữ lập trình khác, có thể truy cập đến từng phần tử của list. Các phần tử của danh sách sẽ được đánh số từ 0.

Ví dụ từ danh sách trên chúng ta sẽ có:

```
>>> a[0]
10
>>> a[3]
8
```

⁵ Tại thời điểm này chúng ta sẽ sử dụng các thuật ngữ danh sách cho **list**, dãy cho **tuple** và dãy thứ tự, vùng số, miền số cho khái niệm **range**. Tất cả các kiểu dữ liệu này đều có thể gọi chung là kiểu dữ liệu tuần tự.

Cách truy cập phần tử của danh sách a với chỉ số k như sau: `a[k]`.

Chú ý quan trọng: Các phần tử của một danh sách (list) có thể có các kiểu dữ liệu khác nhau.

Trong ví dụ sau các phần tử của danh sách a sẽ có các kiểu dữ liệu khác nhau.

```
>>> x = 2; y = "Việt Nam"; z = 3.14
>>> a = [x, y, z]
>>> a
[2, 'Việt Nam', 3.14]
```

2. Các phép toán và phương thức của List

a. Hàm tính độ dài danh sách

Để tính độ dài một danh sách chúng ta dùng hàm số `len()` được thiết lập sẵn trong Python

```
>>> a = [1, 2, 3, 4, 5]
>>> len(a)
5
```

Hàm `len()` có thể dùng cho tất cả các kiểu dữ liệu là dãy các giá trị. Một vài ví dụ khác của hàm `len()`.

```
>>> len(range(9))
9
>>> len([1, 2, 5, 8, 0])
5
```

b. Danh sách rỗng

Chúng ta có thể định nghĩa một danh sách rỗng với cú pháp: `a = []`.

Dãy rỗng sẽ có độ dài 0.

```
>>> a = []
>>> len(a)
0
```

c. Phép + và * danh sách

Phép + sẽ dùng để ghép nối tự nhiên 2 danh sách. Ví dụ:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6, 7, 8, 9, 10]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Phép * với số tự nhiên chính là phép cộng nhiều lần một danh sách. Ví dụ:

```
>>> a = [1, 2, 3]
>>> a*2
[1, 2, 3, 1, 2, 3]
```

d. Xóa một phần tử của danh sách

Muốn xóa phần tử thứ k của danh sách chúng ta dùng lệnh `del`.

```
>> a = [1, 2, 3]
>>> del a[0]
>>> a
[2, 3]
```

Chương trình sau sẽ khởi tạo dãy (list) gồm 100 số từ 1 đến 100.

Tạo_day_1_100.py

```
1. a = []
2. for i in range(1,101):
3.     a = a + [i]
4. print(a)
```

e. Các phương thức của List

Bây giờ chúng ta sẽ cùng xem kỹ các phương thức của kiểu dữ liệu list. Như mọi lớp khác của Python, hàm dir() sẽ hiển thị tất cả các phương thức hiện có của kiểu dữ liệu này.

Trong các phương thức của list, chú ý đến các phương thức được in đậm ở cuối danh sách. Kết quả của lệnh dir() chính là một danh sách dạng List.

```
>>> dir(list)
['_add_', '_class_', '_contains_', '_delattr_',
'_delitem_', '_dir_', '_doc_', '_eq_', '_format_',
'_ge_', '_getattr_', '_getitem_', '_gt_',
'_hash_', '_iadd_', '_imul_', '_init_',
'_init_subclass_', '_iter_', '_le_', '_len_', '_lt_',
'_mul_', '_ne_', '_new_', '_reduce_', '_reduce_ex_',
'_repr_', '_reversed_', '_rmul_', '_setattr_',
'_setitem_', '_sizeof_', '_str_', '_subclasshook_',
'_append', '_clear', '_copy', '_count', '_extend', '_index', '_insert',
'_pop', '_remove', '_reverse', '_sort']
>>>
```

Bảng sau sẽ mô tả nhanh chức năng của các phương thức trên.

Phương thức	Cú pháp	Ý nghĩa
append	a.append(<đối tượng>)	Bổ sung <đối tượng> vào cuối của danh sách như một phần tử.
clear	a.clear()	Xóa dữ liệu, đưa danh sách a thành rỗng.
copy	b = a.copy()	Trả về một danh sách khác có giá trị giống a (một bản sao của a).
extend	a.extend(<đối tượng>)	Mở rộng, bổ sung <đối tượng> vào cuối của danh sách a như một mở rộng tự nhiên.
index	k = a.index(<giá trị>)	Trả về chỉ số index đầu tiên trong danh sách của giá trị <giá trị>.
insert	a.insert(index, <đối tượng>)	Chèn <đối tượng> vào danh sách ở trước vị trí index.
pop	x = a.pop()	Xóa và lấy ra phần tử cuối cùng của danh sách a. Đây chính là thao tác pop của ngăn xếp.
remove	a.remove(<giá trị>)	Xóa đi phần tử đầu tiên của a có giá trị = <giá trị>.

Phương thức	Cú pháp	Ý nghĩa
reverse	<code>a.reverse()</code>	Sắp xếp lại các phần tử của A theo thứ tự ngược lại so với danh sách ban đầu.
sort	<code>a.sort()</code>	Sắp xếp lại danh sách a theo thứ tự tăng dần của giá trị các phần tử.

Chú ý phân biệt sự khác nhau giữa 2 phương thức `append()` và `extend()`.

- Điểm giống nhau là tham số của cả 2 hàm đều là một <đối tượng>. Với `append()` thì đối tượng có thể có kiểu bất kỳ, còn với `extend()` thì đối tượng phải là kiểu tuần tự.

- Sự khác nhau cơ bản ở chức năng: Lệnh `append` sẽ bổ sung <đối tượng> vào như một phần tử ở cuối danh sách. Còn lệnh `extend` sẽ mở rộng danh sách ở cuối, phụ thuộc vào <đối tượng> mà có thể bổ sung một hay nhiều phần tử vào cuối danh sách.

Xét các ví dụ sau. Chú ý: a là danh sách (list) gốc, b là kiểu dữ liệu dãy (tuple) sẽ được học sau (tương tự list), c là một vùng số (kiểu dữ liệu range).

```
>>> a = [1, 2, 3]
>>> b = (4, 5, 6)
>>> c = range(1, 10)
>>> a.append(b)
>>> a
[1, 2, 3, (4, 5, 6)]
>>> a.extend(b)
>>> a
[1, 2, 3, (4, 5, 6), 4, 5, 6]
>>> a.append(c)
>>> a
[1, 2, 3, (4, 5, 6), 4, 5, 6, range(1, 10)]
>>> a.extend(c)
>>> a
[1, 2, 3, (4, 5, 6), 4, 5, 6, range(1, 10), 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Định nghĩa các biến dữ liệu a, b, c.

append b vào cuối a như 1 phần tử.

extend b vào cuối a là 3 phần tử.

append c vào cuối a như 1 phần tử.

extend c vào cuối a là 9 phần tử.

Lệnh sau lỗi vì `extend()` không cho phép có nhiều đối tượng trong cú pháp lệnh của mình.

```
>>> a.extend(10,11,12)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    a.extend(10,11,12)
TypeError: extend() takes exactly one argument (3 given)
```

Giả sử a, b là 2 danh sách dạng List. Bảng sau cho chúng ta lời giải theo các yêu cầu ghép list như sau.

Yêu cầu	Cách thực hiện
Ghép nối a và b, kết quả trả lại vào a.	<code>a = a + b</code> hoặc <code>a.extend(b)</code>
Ghép b vào đầu danh sách a, kết quả trả lại vào a.	<code>a = b + a</code>

3. Lệnh for trên dữ liệu List

Trong bài học 3 chúng ta đã làm quen với lệnh lặp **for** trên vùng các số nguyên của lệnh `range()`. Trong mục này chúng ta sẽ nghiên cứu lệnh **for** một cách tổng thể hơn trong Python. Trước tiên cần biết về khái niệm kiểu dữ liệu tuần tự (iterable).

a. Các kiểu dữ liệu tuần tự

Dữ liệu tuần tự là kiểu dữ liệu đặc trưng bởi 2 tính chất sau:

- Có thể truy cập tuần tự các phần tử của dãy.
- Số lượng phần tử trong dãy là hữu hạn.

Trong Python chúng ta có các kiểu dữ liệu tuần tự sau (trong đó kiểu dữ liệu Tuple sẽ được làm quen ngay trong bài học này).

```
>>> a = [1, 3, 5, 7]
>>> b = 1, 2, 3, 4, 5
>>> c = range(10)
>>> s = "Hà Nội Việt Nam"
>>>
```

a - kiểu danh sách (list)
b - kiểu dãy số (tuple)
c - kiểu vùng số nguyên (range)
s - kiểu chuỗi ký tự (string)

- Tất cả các kiểu dữ liệu tuần tự trên đều có thể sử dụng hàm `len()` để tính độ dài của dãy.
- Để truy cập một phần tử ở vị trí `k` chúng ta có thể viết `a[k]`.
- Chỉ số của các kiểu dữ liệu trên đều đánh số từ 0.

Ví dụ:

```
>>> len(c)
10
>>> s[1]
'à'
>>>
```

b. Lệnh for trên kiểu dữ liệu tuần tự

Lệnh lặp **for** trong Python được định nghĩa trên một kiểu dữ liệu tuần tự bất kỳ với cú pháp như sau:

```
for <item> in <dữ liệu tuần tự>:
    <các lệnh lặp>
```

Khi thực hiện lệnh trên, nhóm lệnh <các lệnh lặp> sẽ được lặp theo độ dài của <dữ liệu tuần tự>, trong mỗi bước lặp, tham biến <item> sẽ lần lượt nhận các giá trị của <dữ liệu tuần tự>.

Chúng ta quan sát một vài ví dụ đơn giản sau.

1. Giả sử ds là danh sách học sinh trong lớp học, chương trình sau sẽ in ra danh sách học sinh lớp học có kèm theo số thứ tự học sinh.

```

1. ds = ["Hà", "Bình", "Huy", "Hoàng", "Quang", "Anh", "Việt",
        "Minh"]           # gán các giá trị ban đầu cho ds
2. count = 0              # biến đánh số thứ tự của danh sách
3. for item in ds:        # bắt đầu vòng lặp chính for
4.     count = count + 1
5.     print(count,item)

```

Kết quả chạy chương trình như sau:

```

1 Hà
2 Bình
3 Huy
4 Hoàng
5 Quang
6 Anh
7 Việt
8 Minh

```

2. Cho trước dãy số tự nhiên A, cần tạo ra dãy số tự nhiên B bao gồm các phần tử là số nguyên tố có trong dãy A. In kết quả dãy A, B ra màn hình.

```

1. A = [2, 19, 30, 45, 100, 101, 103, 257, 890, 1001, 10101]
2. B = []
3. for item in A:
4.     S = 0
5.     for i in range(1,item):
6.         if item % i == 0:
7.             S = S + 1
8.     if S == 1:
9.         B = B + [item]
10. print("Dãy A:")
11. print(A)
12. print("Dãy B:")
13. print(B)

```

Với mỗi phần tử item của dãy A, tính S = số các ước số thực sự của item. Nếu S = 1 thì item là số nguyên tố và bổ sung vào B.

Kết quả chạy chương trình như sau:

```

Dãy A:
[2, 19, 30, 45, 100, 101, 103, 257, 890, 1001, 10101]
Dãy B:
[2, 19, 101, 103, 257]

```

4. Hàm range và chỉ số của dãy List

a. Chỉ số của List

Chỉ số của List, cũng như của một dữ liệu tuần tự bất kỳ sẽ được đánh số từ 0. Chúng ta có thể truy cập trực tiếp đến từng phần tử của danh sách.

Ví dụ:

```

>>> a = [1, "Hà Nội", 3.14]
>>> a[0]
1
>>> len(a)
3
>>>

```

Đặc tính quan trọng nhất của List khác biệt với các kiểu dữ liệu tuần tự khác (như Tuple, String, Range) là chúng ta có thể thay đổi giá trị của từng phần tử của danh sách. Mỗi phần tử của List có thể coi như các biến nhớ và có thể thực hiện các lệnh gán hoặc thay đổi dữ liệu.

Quan sát các lệnh tương tác sau.

```
>>> a[0]="Việt Nam"
>>> a
['Việt Nam', 'Hà Nội', 3.14]
>>> a[1] = [0, 1]
>>> a
['Việt Nam', [0, 1], 3.14]
>>>
```

b. Hàm range

Chúng ta đã được làm quen với hàm range() và kiểu dữ liệu range ở dạng cơ bản nhất. Hàm range() sẽ tạo ra một miền (vùng) các số nguyên và thuộc kiểu dữ liệu tuần tự.

Chúng ta sẽ tìm hiểu sâu hơn hàm số này.

Cú pháp tổng quát của hàm này như sau:

```
range([<start> = 0], <stop>, [<khoảng cách> = 1])
```

Hàm số này có 3 tham số: <start>, <stop> và <khoảng cách> nhưng chỉ có tham số <stop> là thực sự bắt buộc, 2 tham số còn lại là lựa chọn.

Ý nghĩa: Hàm số này sẽ tạo ra một vùng số nguyên, xuất phát từ giá trị <start>, sẽ tăng hoặc giảm dần theo giá trị <khoảng cách> và dừng lại trước khi đạt <stop>.

Mặc định range(n) chính là range(0,n,1).

Ví dụ sau sẽ tạo ra một danh sách các số chẵn không âm và nhỏ hơn n = 30.

```
>>> n = 30
>>> list(range(0,n,2))
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
>>>
```

Muốn thiết lập danh sách (list) từ kết quả của hàm range() chúng ta sử dụng hàm list().

Nếu trong lời gọi hàm chỉ có 2 tham số thì Python hiểu đó là các tham số <start> và <stop>.

```
>>> list(range(5,10))
[5, 6, 7, 8, 9]
```

Nếu <khoảng cách> < 0 thì vùng các số được tạo ra sẽ giảm dần.

Ví dụ:

```
>>> a = range(100,0,-3)
>>> list(a)
[100, 97, 94, 91, 88, 85, 82, 79, 76, 73, 70, 67, 64, 61, 58, 55, 52, 49, 46, 43, 40, 37, 34, 31, 28, 25, 22, 19, 16, 13, 10, 7, 4, 1]
```

c. Một số ứng dụng

Dùng hàm `range()` có thể tạo nhanh các danh sách (list) theo yêu cầu.

1. Cho trước số tự nhiên N , thiết lập một list bao gồm các số $1, 2, 3, \dots, N, N-1, N-2, \dots, 2, 1$.

Lời giải.

```
a = list(range(1, N+1, 1)) + list(range(N-1, 0, -1))
```

2. Cho trước số tự nhiên N , thiết lập list bao gồm N^2 số như sau:

1 2 3 N

2 4 6 2N

.....

N 2N 3N ... N^2

Lời giải.

```
1. A = []
2. N = 8
3. for k in range(1, N+1):
4.     A = A + list(range(k, k*(N+1), k))
5. print(A)
```

5. Kiểu dữ liệu Tuple

Kiểu dữ liệu **tuple** trong Python được gọi là dãy số. Kiểu dãy số (tuple) được định nghĩa trong Python rất gần giống với kiểu dữ liệu danh sách (list). Chúng ta sẽ cùng xem sự giống nhau và khác nhau giữa 2 kiểu dữ liệu quan trọng này trong Python.

Kiểu dữ liệu **tuple** được định nghĩa đơn giản như sau:

<biến nhớ> = (<giá trị 1>, <giá trị 2>, ..., <giá trị N>)

Điểm khác biệt với kiểu dữ liệu list là khi định nghĩa dữ liệu tuple chúng ta dùng ngoặc tròn thay cho ngoặc vuông.

Có thể bỏ ngoặc tròn, chỉ dùng một dãy các giá trị cách nhau bởi dấu phẩy.

Ví dụ sau khởi tạo 2 biến nhớ kiểu dữ liệu dãy số (tuple).

```
>>> a = (1, 2, 3)
>>> b = 12, 3.14, "one", "two", "three"
>>> type(a)
<class 'tuple'>
>>> type(b)
<class 'tuple'>
>>> a
(1, 2, 3)
>>> b
(12, 3.14, 'one', 'two', 'three')
>>>
```

Khác với List, Tuple là kiểu dữ liệu không thể thay đổi được.

Nếu thử thực hiện lệnh gán dữ liệu lên các phần tử của dãy Tuple sẽ có lỗi như sau.

```
>>> a
(1, 2, 3)
>>> a[0] = 10
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    a[0] = 10
TypeError: 'tuple' object does not support item assignment
>>>
```

Tuy nhiên phép + các dữ liệu kiểu dãy vẫn thực hiện được như bình thường. Phép + sẽ ghép nối 2 dãy với nhau.

```
>>> c = a + b
>>> c
(1, 2, 3, 12, 3.14, 'one', 'two', 'three')
>>>
```

Chú ý: Không thể thực hiện phép + một dữ liệu dạng List với một dữ liệu dạng Tuple.
Bây giờ chúng ta cùng xem các phương thức của kiểu dữ liệu Tuple.

```
>>> dir(tuple)
['_add_', '_class_', '_contains_', '_delattr_',
'_dir_', '_doc_', '_eq_', '_format_', '_ge_',
'_getattribute_', '_getitem_', '_getnewargs_', '_gt_',
'_hash_', '_init_', '_init_subclass_', '_iter_',
'_le_', '_len_', '_lt_', '_mul_', '_ne_', '_new_',
'_reduce_', '_reduce_ex_', '_repr_', '_rmul_',
'_setattr_', '_sizeof_', '_str_', '_subclasshook_',
'count', 'index']
>>>
```

Chỉ còn 2 phương thức count() và index() đi với dữ liệu dạng Tuple. Ý nghĩa các hàm này đã được mô tả trong bảng các phương thức của List.

6. Một vài bài toán cụ thể

Chúng ta kết thúc chương này bằng một số bài toán cụ thể.

a. Thiết lập dãy số Fibonacci

Cho trước số tự nhiên N hoặc nhập từ bàn phím số N, cần thiết lập danh sách các phần tử của dãy Fibonacci.

Dãy $F_0, F_1, F_2, \dots$ được gọi là dãy số Fibonacci theo định nghĩa:

$$F_0 = 1, F_1 = 1, F_n = F_{n-1} + F_{n-2} \text{ với } n > 1.$$

Chương trình sau sẽ thiết lập danh sách các số Fibonacci vào biến nhớ F.

Fibonacci.py

```
1. F = [1, 1] # 2 phần tử đầu tiên của dãy Fibonacci
2. N = int(input("Nhập số tự nhiên N: "))
3. a = F[0]; b = F[1];
4. for k in range(2, N+1):
5.     c = a + b
6.     F = F + [c]
7.     a, b = b, c
8. print(F)
```

Trong chương trình trên chú ý các dòng lệnh 5 và 7. Dòng 5 mô tả $c = a + b$. Như vậy chúng ta quy định a, b, c là 3 số hạng liên tiếp của dãy Fibonacci. Dòng 7 sẽ cập nhật lại các giá trị a, b sau khi số c đã được bổ sung vào dãy F.

Ví dụ kết quả chạy chương trình như sau:

```
Nhập số tự nhiên N: 10
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

b. In bảng số nguyên tố

Cần tạo một list bao gồm N số nguyên tố đầu tiên với N số tự nhiên cho trước (hoặc được nhập từ bàn phím).

Day_nguyento.py

```
1. P=[] # danh sách ban đầu rỗng
2. N = int(input("Nhập số tự nhiên N: "))
3. count = 0 # biến đếm số lượng số nguyên tố đã có.
4. m = 2 # giá trị bắt đầu để kiểm tra số nguyên tố
5. while count < N:
6.     S = 0
7.     for k in range(1, m):
8.         if m % k == 0:
9.             S = S + 1
10.    if S == 1:
11.        P = P + [m]
12.        count = count + 1
13.    m = m + 1
14. print(P)
```

c. Nhập một danh sách học sinh từ bàn phím

Chương trình sẽ yêu cầu nhập số N, sau đó sẽ lần lượt nhập N tên học sinh, đưa các tên học sinh này vào một danh sách và in danh sách ra màn hình.

DS_hoc_sinh.py

```
1. HS = [] # danh sách ban đầu rỗng
2. N = int(input("Nhập số học sinh trong lớp: "))
3. for i in range(1, N+1):
4.     s = input("Nhập tên học sinh thứ " + str(i) + ": ")
5.     HS = HS + [s]
6. print("Danh sách học sinh đã nhập:")
7. for i in range(1, N+1):
8.     print(i, HS[i-1])
```


Ví dụ kết quả chạy của chương trình:

```
Nhập số học sinh trong lớp: 4
Nhập tên học sinh thứ 1: Hà
Nhập tên học sinh thứ 2: Bình
Nhập tên học sinh thứ 3: Quang
Nhập tên học sinh thứ 4: Nguyễn Văn Thuật
Danh sách học sinh đã nhập:
1 Hà
2 Bình
3 Quang
4 Nguyễn Văn Thuật
```



Câu hỏi - bài tập

1. Lệnh `list(range(-3,3))` trả lại giá trị nào?

- a) `[-3, -2, -1, 0, 1, 2, 3]`
- b) `[-3, -2, -1, 0, 1, 2]`
- c) `[-2, -1, 0, 1, 2]`
- d) `[-3, 3]`

2. Kết quả đoạn chương trình sau là gì?

```
>>> A = []
>>> A.append([1,2,3])
>>> print(A)
```

- a) `[1, 2, 3]`
- b) `1, 2, 3`
- c) `[[1, 2, 3]]`
- d) `([1, 2, 3])`

3. Kết quả của đoạn chương trình sau là gì?

```
>>> X = []
>>> X.append(1)
>>> X.append(5)
>>> print(X)
```

- a) `[]`
- b) `1, 5`
- c) `[1, 5]`
- d) `1 5`

4. Lệnh sau `list(range(-3,3,3))` trả lại giá trị nào?

- a) `[-3, 3]`
- b) `[-3, 0, 3]`
- c) `[-3, 0]`

5. Lệnh sau `list(range(-3))` trả lại giá trị nào?

- a) `[-3, 3]`
- b) `[-3, 0, 3]`
- c) `[-3, 0]`
- d) `[]`

6. Hàm số sau thực hiện công việc gì?

```
>>> def multi(start, stop):  
    p = 1  
    for i in range(start, stop):  
        p = p * i  
    return p  
>>> multi(1, 6)
```

Kết quả các lệnh trên là bao nhiêu?

- a) 24
- b) 100
- c) 120
- d) 720

7. Kết quả của các lệnh sau như thế nào?

```
>>> A = [9, 1, 15, 0, "abc", 2, "de"]  
>>> A.sort()
```

- a) `["abc", "de", 0, 1, 2, 9, 15]`
- b) `[0, 1, 2, 9, 15, "abc", "de"]`
- c) Thông báo lỗi.
- d) `[0, 1, 2, 9, 15]`

8. Viết chương trình nhập số tự nhiên n từ bàn phím, in và liệt kê danh sách các ước số thực sự của n như một danh sách (list).

Ví dụ nếu $n = 15$ thì kết quả dãy (list) là `[1, 3, 5]`.

9. Hàm số sau thực hiện công việc gì?

```
1. def listover(A):  
2.     B = []  
3.     for item in range(len(A)-1, -1, -1):  
4.         B.append(A[item])  
5.     return B
```

10. Viết chương trình thực hiện công việc sau:

- Nhập từ bàn phím số tự nhiên n .
- Nhập lần lượt n số tự nhiên tạo thành dãy $a_1, a_2, \dots, a_n$. Giả sử dãy này gọi là A .
- Đưa ra dãy con $b_1, b_2, \dots, b_k$ của dãy A bao gồm các phần tử là nguyên tố của dãy này.

11. Nhập dãy A bao gồm n phần tử từ bàn phím và tính tổng các phần tử của A .

12. Viết chương trình nhập số tự nhiên n từ bàn phím và tính các giá trị sau.

a) Tính tổng $1^2 + 2^2 + 3^2 + \dots + n^2$.

b) Tính tổng $1 + 1/2 + 1/3 + \dots + 1/n$.

c) Tính tổng $\frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \dots + \frac{1}{(n-1)n}$

13. Cho trước 2 dãy A, B. Dãy C được thiết lập bằng cách lần lượt lấy các phần tử từ A, B và đưa vào C.

Ví dụ:

A = [0, 2, 4, 6, 8, 10].

B = [1, 3, 5, 7, 9, 11].

Thì C = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11].

Viết một hàm với 2 tham số đầu vào là A, B và trả lại dãy C như trên.

14. Cho trước dãy A. Dãy B được thiết lập bằng cách tách các phần tử là số (số nguyên hoặc số thực) từ dãy A. Ví dụ:

A = [1.2, "one", 0, 15, "A", "B", "C", 1, 50, 4.56, "C"].

Khi đó dãy B sẽ là:

B = [1.2, 0, 15, 1, 50, 4.56].

Viết một hàm số với tham số đầu vào là dãy A và kết quả trả lại dãy B.

15. Viết chương trình nhập một chuỗi ký tự từ bàn phím. Kết quả là một dãy các ký tự của chuỗi này. Ví dụ:

Chuỗi nhập vào: Việt Nam.

Kết quả ra: ["V", "i", "ệ", "t", " ", "N", "a", "m"].

16. Viết chương trình nhập số tự nhiên n và thiết lập:

- Dãy A bao gồm các số tự nhiên lẻ nhỏ hơn n .

- Dãy B bao gồm các số tự nhiên chẵn nhỏ hơn n .

17. Cho trước 2 dãy số A, B. Dãy C được tạo ra theo quy tắc sau:

Giá trị số phần tử của A là n và nhỏ hơn số phần tử của B. Trường hợp ngược lại làm tương tự.

- n phần tử đầu tiên của C là tổng các số hạng tương ứng của A, B.

- Các phần tử còn lại lấy từ B.

Viết thủ tục với 2 tham số đầu vào là A, B và in ra màn hình dãy C.

18. Viết các hàm số với đầu vào là dãy A (bao gồm số và chuỗi ký tự).

a) Tính tổng các phần tử là số trong dãy A.

b) Đếm số các phần tử là chuỗi ký tự từ dãy A.

c) Đưa ra kết quả là 2 dãy B, C. Dãy B bao gồm các phần tử là số của A, dãy C bao gồm các phần tử là chuỗi ký tự của A.

19. Viết một hàm số thay thế cho phương thức **sort()** của dãy (list). Tham số đầu vào là dãy A bao gồm các phần tử là số (nguyên hoặc thực). Hàm trả lại là một dãy bao gồm các số đã được sắp xếp theo thứ tự tăng dần các phần tử từ A.

20. Viết một hàm số tổng quát hơn phương thức **sort()** của dãy (list) như sau:

- Đầu vào là dãy A bao gồm có thể là số hoặc ký tự.

- Đầu ra là dãy B thu được từ dãy A bằng cách sắp xếp các phần tử là số theo thứ tự tăng dần, còn các vị trí là ký tự thì giữ nguyên.

Ví dụ:

A = [1.2, "one", 0, 15, "A", "B", "C", 1, 50, 4.56, "C"].

B = [0, "one", 1, 1.2, "A", "B", "C", 4.56, 15, 50, "C"].

Đặt tên hàm số này là **general_sort()**.

21. Cho trước 2 dãy (list) A, B và số tự nhiên k, $0 \leq k < \text{len}(A)$. Viết hàm số trả lại dãy (list) thu được từ A sau khi chèn dãy B vào vị trí của chỉ số k. Ví dụ:

A = [0, 1, 2, 3, 4, 5], B = ["a", "b", "c"], k = 1 thì dãy (list) thu được sẽ là:

C = [0, "a", "b", "c", 1, 2, 3, 4, 5].

22. Cho trước dãy (list) a bao gồm các số nguyên, cho trước số nguyên k. Biết rằng dãy (list) trong Python có 2 phương thức là **count()** và **remove()** có ý nghĩa như sau:

a.count(<giá trị>) sẽ trả lại số lần tìm thấy của <giá trị> trong a, nếu không tìm thấy sẽ trả lại 0.

a.remove(<giá trị>) sẽ xóa phần tử đầu tiên có <giá trị> trong a, nếu không tìm thấy phần tử có <giá trị> sẽ báo lỗi.

Viết chương trình xóa tất cả các phần tử trong dãy a có giá trị k. Chú ý viết chương trình không được để gặp lỗi.

23. Cho trước dãy (list) a bao gồm các số nguyên, cho trước 2 số nguyên k, m. Viết chương trình thay thế tất cả các phần tử của a có giá trị k bằng giá trị m. Gợi ý: Sử dụng phương thức **index()** của list trong Python.

24. Cho trước 2 dãy (list) a, b. Viết chương trình thiết lập dãy c bao gồm các phần tử lấy lần lượt từ phần cuối của a, b. Gợi ý: Sử dụng phương thức **pop()** của list trong Python.

25. Hãy viết một lệnh để tạo dãy a bao gồm m số 0 và n số 1 với m, n là các số tự nhiên cho trước.

26. Cho trước dãy (list) A. Lệnh sau thực hiện công việc gì (với i, j là các chỉ số bất kỳ)?

A[i], A[j] = A[j], A[i]

27. Cho trước dãy (list) A. Viết chương trình tạo ra dãy B bao gồm các phần tử lấy từ A nhưng theo thứ tự ngược lại. Chú ý yêu cầu giữ nguyên dãy A không thay đổi.

28. Cho trước 2 dãy số A, B có số phần tử bằng nhau và đều là các số tự nhiên. Viết chương trình tạo dãy C cũng có số phần tử bằng A, B và các phần tử của C là ước số chung lớn nhất của các phần tử tương ứng của A, B.

Chương 7. List của List. Mảng nhiều chiều

Mục tiêu:

- List của List hay dữ liệu mảng nhiều chiều.
- Các tính chất nâng cao hơn của danh sách.

Nội dung chính

Trong chương này chúng ta sẽ cùng làm quen với các tính năng nâng cao liên quan đến kiểu dữ liệu List, một trong tính năng nâng cao quan trọng nhất là khả năng khai báo List của List và điều này sẽ dẫn đến khái niệm mảng 2 hoặc nhiều chiều. Trong Python không có khái niệm mảng nhiều chiều nhưng chúng ta hoàn toàn có thể làm việc với các dữ liệu dạng này thông qua List.

1. Các vòng lặp lồng nhau

Chúng ta đã được làm quen với lệnh lặp **for** trong các bài học trước. Trong chương này chúng ta sẽ quay lại vấn đề này để tìm hiểu sâu hơn các tính năng của lệnh **for** với kiểu dữ liệu List.

Cho một dãy số được gán trước trong danh sách A. Cần sắp xếp dãy A này lại theo thứ tự tăng dần. Đoạn chương trình sau bao gồm 2 vòng lặp **for** lồng nhau sẽ thực hiện được bài toán trên.

```
for i in range(len(A) - 1):  
    for j in range(i+1, len(A)):  
        if A[i] > A[j]:  
            A[i], A[j] = A[j], A[i]
```

2 vòng for lồng nhau tại đây.

Đổi vị trí 2 phần tử.

Chương trình hoàn chỉnh như sau.

SX_day_so.py

```
1. A = [12, 1, 0, 23, 15, 6, 18, 40, 10, 55, 3]  
2. print("Dãy số gốc:")  
3. print(A)  
4. for i in range(len(A) - 1):  
5.     for j in range(i+1, len(A)):  
6.         if A[i] > A[j]:  
7.             A[i], A[j] = A[j], A[i]  
8. print("Dãy số sau khi sắp xếp:")  
9. print(A)
```

Chúng ta sẽ tìm cách thể hiện các ma trận số trên dữ liệu của Python. Một ma trận là bảng có dạng sau:

$a_{11}, a_{12}, \dots, a_{1n}$

$a_{21}, a_{22}, \dots, a_{2n}$

.....

$a_{m1}, a_{m2}, \dots, a_{mn}$

Ma trận này có $m \times n$ phần tử, bao gồm m hàng, mỗi hàng n cột.

Cho trước 2 số tự nhiên m, n , cần in ra bảng - ma trận sau:

(1, 1) (1, 2) ... (1, n)

(2, 1) (2, 2) ... (2, n)

.....

(m , 1) (m , 2) ... (m , n)

Đoạn chương trình sẽ có dạng như sau.

```
m = 4; n = 6
for i in range(m):
    for j in range(n):
        print("(", i+1, ", ", j+1, ")", sep="", end = " ")
        print()
```

Lệnh này thiết lập xuống dòng.

Kết quả của chương trình trên như sau:

```
(1, 1) (1, 2) (1, 3) (1, 4) (1, 5) (1, 6)
(2, 1) (2, 2) (2, 3) (2, 4) (2, 5) (2, 6)
(3, 1) (3, 2) (3, 3) (3, 4) (3, 5) (3, 6)
(4, 1) (4, 2) (4, 3) (4, 4) (4, 5) (4, 6)
```

2. List của List

Chúng ta đã biết cấu trúc của dữ liệu List là một danh sách hữu hạn các phần tử nhưng có thể có các kiểu dữ liệu khác nhau.

Ví dụ:

```
>>> a = [1, "Bút chì", 3.14]
>>> type(a[0])
<class 'int'>
>>> type(a[1])
<class 'str'>
```

Trong ví dụ trên danh sách a có 3 phần tử và mỗi phần tử là một kiểu dữ liệu khác nhau. Do vậy về nguyên tắc, a có thể có các phần tử thuộc kiểu dữ liệu List.

Chúng ta sẽ bổ sung thêm một phần tử nữa vào cuối danh sách, phần tử này thuộc kiểu List.

```
>>> a.append([0, 1])
>>> a
[1, 'Bút chì', 3.14, [0, 1]]
```

Phần tử cuối cùng của a là $[0, 1]$ là một giá trị kiểu List.

```
>>> a[3]
[0, 1]
>>> a[3][0] ← Đây là phần tử thứ nhất (chỉ số 0)
0              của a[3]
```

Như vậy trong mô hình List of List thì ý nghĩa của chỉ số kép như sau:

$A[i][j]$ $\longrightarrow$ Phần tử thứ j của List $A[i]$

Mô hình dữ liệu List của List có khuôn dạng như sau:

$A = [<list1>, <list2>, \dots, <listN>]$

3. Mô tả ma trận trong Python

Chúng ta sẽ mô tả dữ liệu trong Python để hỗ trợ cho mô hình ma trận sau:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Ma trận này được mô tả bằng kiểu dữ liệu List của List như sau.

$A = [[a_{11}, a_{12}, \dots, a_{1n}], [a_{21}, a_{22}, \dots, a_{2n}], \dots, [a_{m1}, a_{m2}, \dots, a_{mn}]]$

Để hiện hình ảnh ma trận trên màn hình chúng ta dùng đoạn chương trình sau.

```
1. for i in range(m):
2.     for j in range(n):
3.         print(A[i][j], end = " ")
4.     print()
```

Sau đây là một chương trình hoàn chỉnh, yêu cầu nhập từ bàn phím các số tự nhiên m, n , sau đó nhập từng phần tử của ma trận và in kết quả ma trận ra màn hình.

Nhap_ma_tran.py

```
1. a = []
2. m = int(input("Nhập số tự nhiên m: "))
3. n = int(input("Nhập số tự nhiên n: "))
4. for i in range(m):
5.     print("Chuẩn bị nhập ma trận hàng thứ ", i+1, ":")
6.     b = []
7.     for j in range(n):
8.         x = int(input("Nhập phần tử thứ "+str(j+1)+" : "))
9.         b = b + [x]
10.    a.append(b)
11. print("Ma trận A đã nhập xong.")
12. # in ma trận ra màn hình
13. for i in range(m):
14.     for j in range(n):
15.         print(a[i][j], end = " ")
16.     print()
```

Chú ý: Trong ví dụ trên các phần tử của ma trận được nhập vào là số nguyên, tuy nhiên có thể nhập các giá trị bất kỳ.

Kết quả của chương trình trên có thể như sau.

```
Nhập số tự nhiên m: 3
Nhập số tự nhiên n: 3
Chuẩn bị nhập ma trận hàng thứ 1 :
Nhập phần tử thứ 1: 1
Nhập phần tử thứ 2: 2
Nhập phần tử thứ 3: 3
Chuẩn bị nhập ma trận hàng thứ 2 :
Nhập phần tử thứ 1: 4
Nhập phần tử thứ 2: 5
Nhập phần tử thứ 3: 6
Chuẩn bị nhập ma trận hàng thứ 3 :
Nhập phần tử thứ 1: 7
Nhập phần tử thứ 2: 8
Nhập phần tử thứ 3: 9
Ma trận A đã nhập xong.
1 2 3
4 5 6
7 8 9
```

4. List of List - dữ liệu có cấu trúc

a. Dữ liệu có cấu trúc

Trong các phần trên chúng ta đã làm quen với kiểu dữ liệu danh sách thông thường và dữ liệu ma trận. Các kiểu dữ liệu này được gọi là dữ liệu có cấu trúc. Nói một cách đơn giản dữ liệu có cấu trúc là một bộ dữ liệu được sắp xếp và ghép lại theo một quy luật nào đó. Chúng ta sẽ làm quen với một số cấu trúc dữ liệu đơn giản sử dụng List of List.

1. Chúng ta muốn lập một danh sách bạn bè, nhưng ngoài thông tin chính là tên bạn chúng ta còn muốn lưu thêm thông tin tên lớp của bạn đó. Như vậy mẫu dữ liệu có cấu trúc sẽ là <tên học sinh>, <lớp học>. Đoạn chương trình sau mô tả định nghĩa dữ liệu này

```
>>> Data = [{"Binh", "10A"}, {"An", "10B"}, {"Vinh", "11B"},
             {"Thinh", "9C"}]
>>> Data
[["Binh", "10A"], ["An", "10B"], ["Vinh", "11B"], ["Thinh", "9C"]]
>>>
```

Giả sử chúng ta muốn danh sách học sinh và thông tin lớp học tương ứng trong bộ dữ liệu trên, chương trình có thể như sau:

```
>>> for i in range(len(Data)):
    print(i+1, Data[i][0], Data[i][1])
1 Binh 10A
2 An 10B
3 Vinh 11B
4 Thinh 9C
>>>
```

2. Ví dụ tiếp theo, giả sử em cần lưu một danh sách các tổ học tập của lớp em, mỗi tổ có tên riêng và danh sách các bạn trong tổ. Ví dụ sau thiết lập danh sách 4 tổ với tên gọi Xanh, Đỏ, Tím, Vàng. Chú ý là số lượng học sinh trong mỗi tổ không bằng nhau.

```
A = [{"Xanh", "Bình", "Hà", "Sơn", "Thuận"},  
      ["Đỏ", "Hiền", "Công", "Thành"],  
      ["Tím", "Thành B", "Huy", "Dương", "Trần", "Cường"],  
      ["Vàng", "Hoa", "Hồng", "Anh", "Thịnh",  
      "Hiệp", "Hoàng", "Yến"]]
```

Bây giờ chúng ta muốn in danh sách các nhóm, mỗi nhóm cần in riêng biệt. Ví dụ muốn in ra dưới dạng:

Nhóm Xanh:

Bình, Hà, Sơn, Thuận

Tương tự sẽ in ra tên các bạn trong nhóm Đỏ, Tím, Vàng.

Chương trình sau sẽ thực hiện công việc này.

In_danh_sach_nhom.py

```
1. A = [{"Xanh", "Bình", "Hà", "Sơn", "Thuận"},  
2.     ["Đỏ", "Hiền", "Công", "Thành"],  
3.     ["Tím", "Thành B", "Huy", "Dương", "Trần", "Cường"],  
4.     ["Vàng", "Hoa", "Hồng", "Anh", "Thịnh",  
      "Hiệp", "Hoàng", "Yến"]]  
5. for i in range(len(A)):  
6.     print("Nhóm", A[i][0], ":", sep = "")  
7.     print("        ", end = "")  
8.     for j in range(1, len(A[i])):  
9.         print(A[i][j], " ", sep = "", end = "")  
10.    print()
```

Kết quả in ra sẽ như sau:

Nhóm Xanh:

Bình, Hà, Sơn, Thuận,

Nhóm Đỏ:

Hiền, Công, Thành,

Nhóm Tím:

Thành B, Huy, Dương, Trần, Cường,

Nhóm Vàng:

Hoa, Hồng, Anh, Thịnh, Hiệp, Hoàng, Yến,

b. Lệnh gán đồng thời với List

Chúng ta đã biết có thể gán trực tiếp dữ liệu cho List như sau.

```
a = [1, 2, 3]
```

Ngược lại có thể gán tham biến đồng thời với dữ liệu List như sau, ví dụ:

```
x, y, z = a
```

Tổng quát nếu a là dữ liệu kiểu List có N phần tử thì có thể thực hiện lệnh gán đồng thời như sau:

```
<biến nhớ 1>, <biến nhớ 2>, ..., <biến nhớ N> = a
```

Xét ví dụ:

```
>>> a = [1, 2, 3]
>>> x, y, z = a
>>> x
1
>>> y
2
>>> z
3
>>>
```

Gán đồng thời và đọc lập các giá trị x, y, z cho các phần tử của a.

c. Lệnh for với List of List

Chúng ta nhớ lại định nghĩa lệnh **for** trên kiểu dữ liệu List (hay Tuple hoặc String) đã nêu trong bài 6 như sau:

```
for <item> in <dữ liệu tuần tự>:
    <các lệnh lặp>
```

Lệnh này có thể mở rộng cho trường hợp dữ liệu List of List rất hay như sau. Nếu các phần tử của một <list> là kiểu dữ liệu List có N phần tử thì lệnh lặp **for** trên <list> có thể như sau.

```
for <item1>, <item2>, ..., <itemN> in <list>:
    <các lệnh lặp>
```

Ý nghĩa lệnh trên như sau: Tại mỗi vòng lặp các biến nhớ <item1>, <item2>, ..., <itemN> sẽ lần lượt nhận các giá trị tương ứng của danh sách là phần tử của <list> để thực hiện các lệnh của vòng lặp.

Lệnh **for** tổng quát trên với dữ liệu List có rất nhiều ứng dụng, chúng ta cùng tìm hiểu qua một số ví dụ.

1. Giả sử có một danh sách dữ liệu các bạn học sinh trong lớp, mỗi bạn học sinh sẽ có 3 dữ liệu sau:

- Tên học sinh.
- Năm sinh.
- Quê quán.

Chương trình sau sẽ in ra danh sách học sinh. Sử dụng vòng lặp **for** tổng quát trên, chương trình sẽ viết rất đơn giản.

```
1. A = [{"Bình", 2010, "Hà Nội"},
2.      {"Sơn", 2011, "Thái Bình"},
3.      {"Quang", 2008, "Quảng Nam"}]
4. for ten, namsinh, que in A:
5.     print("Tên HS:", ten, "Năm sinh:", namsinh, "Quê quán:", que)
```

Kết quả chương trình trên sẽ như sau:

```
Tên HS: Bình Năm sinh: 2010 Quê quán: Hà Nội
Tên HS: Sơn Năm sinh: 2011 Quê quán: Thái Bình
Tên HS: Quang Năm sinh: 2008 Quê quán: Quảng Nam
```


2. Trong ví dụ này, điểm số của học sinh trong lớp được lưu trong bộ dữ liệu list có khuôn dạng như sau, mỗi phần tử của danh sách sẽ bao gồm 3 bộ dữ liệu:

- Tên học sinh.
- Các điểm số hệ số 1.
- Các điểm số hệ số 2.

Trong đó các điểm hệ số 1 và 2 được lưu trong một dãy list riêng.

Ví dụ có thể thiết lập bộ dữ liệu điểm số như vậy bằng lệnh sau:

```
D = [{"Hà", [6, 7, 9], [7, 3, 10, 6]},  
      ["Đức", [10, 9, 10, 8], [6, 8, 7]],  
      ["Ngọc", [7, 10, 9], [10, 6, 5, 5, 9]]]
```

Yêu cầu in bảng điểm của lớp học trên.

Sau đây là chương trình in điểm số của cả lớp, hiện lần lượt từng cá nhân, nhưng nhóm theo từng loại điểm.

In_diem_so.py

```
1. D = [{"Hà", [6, 7, 9], [7, 3, 10, 6]},  
2.     ["Đức", [10, 9, 10, 8], [6, 8, 7]],  
3.     ["Ngọc", [7, 10, 9], [10, 6, 5, 5, 9]]]  
4. for ten, diem1, diem2 in D:  
5.     print(ten)  
6.     print("    Hệ số 1: ", end = " ")  
7.     for diem in diem1:  
8.         print(diem, end = " ")  
9.     print()  
10.    print("    Hệ số 2: ", end = " ")  
11.    for diem in diem2:  
12.        print(diem, end = " ")  
13.    print()
```

Chúng ta chú ý đến dòng lệnh 4, ở đó lệnh lặp **for** được gọi trên bộ dữ liệu D, với mỗi vòng lặp cả 3 biến nhớ ten, diem1 và diem2 được đồng thời gán với dữ liệu tương ứng của tên học sinh, các điểm hệ số 1 và các điểm hệ số 2.

Tại các dòng lệnh 7 và 11, chương trình sẽ in chi tiết các điểm hệ số 1 và hệ số 2 trên một dòng ngang.

Kết quả của chương trình như sau:

Hà	Hệ số 1:	6	7	9	
	Hệ số 2:	7	3	10	6
Đức	Hệ số 1:	10	9	10	8
	Hệ số 2:	6	8	7	
Ngọc	Hệ số 1:	7	10	9	
	Hệ số 2:	10	6	5	5 9

5. Chỉ số âm và vùng chỉ số (Slicing)

a. Chỉ số âm của List (Tuple, String, Range)

Chúng ta đã biết về chỉ số truy cập từng phần tử của dữ liệu danh sách tuần tự (List, Tuple, String, Range). Các phần tử của danh sách sẽ được đánh chỉ số từ 0. Bên cạnh các chỉ số thông thường đó, danh sách còn hỗ trợ cách đánh chỉ số âm.

Giả sử danh sách a có các phần tử như sau, đánh số từ a_1 đến a_6 .

$a = [a_1, a_2, a_3, a_4, a_5, a_6]$

Khi đó quy định chỉ số của danh sách a như sau.

	a_1	a_2	a_3	a_4	a_5	a_6
Chỉ số dương	0	1	2	3	4	5
Chỉ số âm	-6	-5	-4	-3	-2	-1

Quan sát các ví dụ sau để hiểu hơn về chỉ số âm.

```
>>> a = ["One", "Two", "Three", 4, 5, 6]
>>> a[-6]
'One'
>>> a[-1]
6
```

Theo quy định trên ta có công thức liên hệ giữa chỉ số âm và chỉ số mặc định của list.

$a[0] = a[-len(a)]$

$a[1] = a[-len(a) + 1] = a[1 - len(a)]$

$a[k] = a[k - len(a)]$

b. Vùng chỉ số (Slicing)

Vùng chỉ số là một miền liên tục chỉ số được xác định trên các kiểu dữ liệu tuần tự. Giả sử A là một biến nhớ dạng list. Khi đó cú pháp sau sẽ tạo một vùng dữ liệu mới, dựa trên biểu thức vùng chỉ số. Cú pháp này được gọi là **Slicing**.

```
newlist = A[<start> : <stop>]
```

Vùng chỉ số của lệnh trên sẽ tạo một dữ liệu dạng list mới, có các phần tử lấy của A, tính từ vị trí chỉ số <start> và kết thúc trước chỉ số <stop>. Như vậy lệnh trên sẽ tạo một danh sách mới (có tên là newlist), bao gồm $A[<start>]$, $A[<start>+1]$, ..., $A[<stop> - 1]$.

Ví dụ:

```
a = ["One", "Two", "Three", 4, 5, 6]
>>> a[0 : 3]
['One', 'Two', 'Three']
>>> a[2 : ]
['Three', 4, 5, 6]
>>> a[ : 4]
['One', 'Two', 'Three', 4]
>>> a[ : ]
['One', 'Two', 'Three', 4, 5, 6]
>>>
```

Nếu <stop> không có thì mặc định hiểu là len(a).

Nếu <start> không có thì mặc định hiểu là 0.

Nếu cả <start> và <stop> đều không có thì hiểu là toàn bộ list.

- Nếu <start> không có thì mặc định hiểu là 0.
- Nếu <stop> không có thì mặc định hiểu là len(a).
- Nếu cả <start> và <stop> đều không có thì hiểu là toàn bộ list.

Chúng ta có thể dùng vùng chỉ số để gán trực tiếp dữ liệu cho một miền các giá trị của dãy số dạng list.

$A[\text{<start> : <stop>}] = \text{<dãy các giá trị>}$

Ví dụ:

<pre>>>> a = [0, 1, 2, 3, 4, 5] >>> a[0 : 2] = "a", "b" >>> a ['a', 'b', 2, 3, 4, 5]</pre>	—————> Lệnh này gán đồng thời 2 giá trị "a", "b" cho các phần tử a[0] và a[1].
---	--

Chú ý quan trọng:

Nếu các giá trị <start> hay <stop> có giá trị âm thì phải được mặc định hiểu sau khi biến đổi chỉ số này sang giá trị dương tương ứng (hoặc có thể đổi tất cả về các chỉ số âm cũng được). Ví dụ:

```
>>> a[-6 : -1] # tương đương a[0 : 5]
['One', 'Two', 'Three', 4, 5]
>>> a[-1:0]    # tương đương a[5 : 0] hoặc a[-1 : -6]
[]
>>> a[0 : -1]  # tương đương a[0 : 5] hoặc a[-6 : -1]
['One', 'Two', 'Three', 4, 5]
>>> a[4 : -2]  # tương đương a[4 : 4] hoặc a[-2 : -2]
[]
```

Lệnh vùng chỉ số (Slicing) còn có cú pháp mở rộng như sau:

$\text{newlist} = A[\text{<start> : <stop> : <increment>}]$

Lệnh này trả lại một list mới, lấy các phần tử của A, bắt đầu từ chỉ số <start>, theo quãng tăng <increment> và kết thúc trước vị trí <stop>. Nếu không ghi rõ giá trị <increment> thì mặc định hiểu là 1.

Chú ý quan trọng:

Nếu <increment> lớn hơn 0 thì mặc định <start> = 0 và <stop> = len().

Nếu <increment> nhỏ hơn 0 thì mặc định <start> = -1 và <stop> = - len() - 1.

Ví dụ:

<pre>>>> a = ["One", "Two", "Three", 4, 5, 6] >>> a[0 : 5: 2] ['One', 'Three', 5] >>> a[: : -1] [6, 5, 4, 'Three', 'Two', 'One'] >>> a[0 : 5 : -1] [] >>> a[: : 2] = [1, 1, 1] >>> a [1, 'Two', 1, 4, 1, 6]</pre>	—————> a[: : -1] sẽ tạo list với các phần tử xếp theo chiều ngược lại so với list ban đầu. —————> a[: : 2] là dãy 3 phần tử với chỉ số 0, 2, 4 được gán trực tiếp bởi các số 1.
---	---

6. Liên kết tham biến của List

Trong mục này chúng ta sẽ quan sát và tìm hiểu về cách liên kết tham chiếu của dữ liệu kiểu list và so sánh với những kiến thức đã biết, có nhiều điều đặc biệt về dữ liệu list so với các dữ liệu chuẩn đã học. Trước tiên cùng quan sát các ví dụ.

6.1. Các ví dụ tham chiếu liên kết dữ liệu chuẩn và list

Trong ví dụ 1, biến nhớ x được khởi tạo và gán = 5. Sau đó gán tiếp y = x. Bây giờ x và y sẽ cùng chỉ đến đối tượng số là 5. Nhưng khi gán tiếp x = 3 thì giá trị của x và y sẽ khác nhau. Toán tử is dùng để kiểm tra 2 đối tượng số có giống nhau hay không.

Ví dụ 1. Tham chiếu biến nhớ dữ liệu chuẩn.

```
1. >>> x = 5
2. >>> y = x
3. >>> x = 3
4. >>> y
5. 5
6. >>> x is y
7. False
```

Kết quả x is y = False, vậy x và y nhận các giá trị khác nhau.

Bây giờ chúng ta thực hiện các bước tương tự như ví dụ 1 nhưng đối với 2 biến nhớ dạng list là xlist và y list. Quan sát sau khi thực hiện các bước tương tự như ví dụ 1 thì kết quả trả lại là xlist và ylist vẫn giống nhau!! Vì sao lại như vậy?

Ví dụ 2. Tham chiếu list.

```
1. >>> xlist = [1, 2]
2. >>> ylist = xlist
3. >>> xlist[0] = 10
4. >>> ylist
5. [10, 2]
6. >>> xlist is ylist
7. True
```

Thay đổi giá trị một thành phần của xlist.

Kết quả xlist is ylist = True, vậy xlist và ylist giống nhau. Hoàn toàn khác với ví dụ 1.

Ví dụ tiếp theo kiểm tra biến nhớ tổng thể có thay đổi không nếu truyền giá trị cho hàm số. Chúng ta đã biết trong hàm số mọi biến nhớ đều được hiểu như biến địa phương và không ảnh hưởng đến bên ngoài hàm số. Minh họa điều này trong ví dụ 3.

Ví dụ 3. Biến địa phương bên trong hàm số.

```
1. >>> def f(x):
2.     x = x + 1
3.     return x
4. >>> a = 3
5. >>> f(a)
6. 4
7. >>> a
8. 3
```

Biến nhớ a không thay đổi giá trị khi ra khỏi hàm f(x).

Trong ví dụ trên, biến nhớ a được gán giá trị 3, và được truyền vào hàm số f(x). Bên trong hàm số này, biến nhớ a được tăng lên 1 đơn vị, nhưng khi ra khỏi hàm số, biến nhớ a vẫn không thay đổi.

Bây giờ chúng ta thực hiện tương tự ví dụ trên nhưng với các biến nhớ dạng list. Kết quả lại hoàn toàn khác!!

Biến nhớ dạng list `xlist` ra khỏi hàm số `flist()` đã bị thay đổi theo đúng những gì đã thực hiện bên trong hàm số này. Như vậy biến `xlist` sẽ không thể là biến địa phương bên trong hàm số `flist()`.

Ví dụ 4. List bên trong hàm số.

```
1. >>> def flist(x):
2.     x[0] = 2*x[0]
3.     return x
4. >>> xlist = [5, 5]
5. >>> flist(xlist)
6. [10, 5]
7. >>> xlist
8. [10, 5]
```

Giá trị của `xlist` trước khi chạy hàm `flist()`

Giá trị của `xlist` sau khi chạy hàm `flist()`, đã thay đổi.

Từ các ví dụ trên chúng ta thấy rõ sự khác biệt giữa tham chiếu biến kiểu dữ liệu chuẩn hoàn toàn khác biệt với biến kiểu dữ liệu list. Trong phần tiếp theo chúng ta sẽ cùng giải thích nguyên nhân này.

Để hiểu rõ hơn sự khác biệt giữa kiểu dữ liệu list và các kiểu dữ liệu chuẩn, chúng ta cùng quan sát tiếp theo 2 ví dụ nữa. Với 2 ví dụ này, biểu hiện của biến nhớ dạng list đã hoàn toàn giống như các biến nhớ dữ liệu chuẩn. Xem kỹ hơn ví dụ để thấy sự khác biệt này.

Ví dụ 5 cần so sánh với ví dụ 2 để thấy sự khác biệt.

Ví dụ 5. Tham chiếu list.

```
1. >>> xlist = [1, 2]
2. >>> ylist = xlist
3. >>> xlist = xlist + xlist
4. >>> xlist is ylist
5. False
6. >>> xlist
7. [1, 2, 1, 2]
8. >>> ylist
9. [1, 2]
```

Tại đây, `xlist` và `ylist` cùng chỉ vào một đối tượng list.

Tại đây, `xlist` được gán một đối tượng list mới.

Bây giờ `xlist` và `ylist` đã khác nhau.

Ví dụ tiếp theo, biến nhớ dạng list là `xlist` sau khi được truyền vào hàm số `sflist()`, khi ra ngoài vẫn không thay đổi.

Ví dụ 6 cần so sánh với ví dụ 4 để thấy sự khác biệt.

Ví dụ 6. List bên trong hàm số.

```
1. >>> def dflist(x):
2.     x = x + x
3.     return x
4. >>> xlist = [10, 5]
5. >>> dflist(xlist)
6. [10, 5, 10, 5]
7. >>> xlist
8. [10, 5]
```

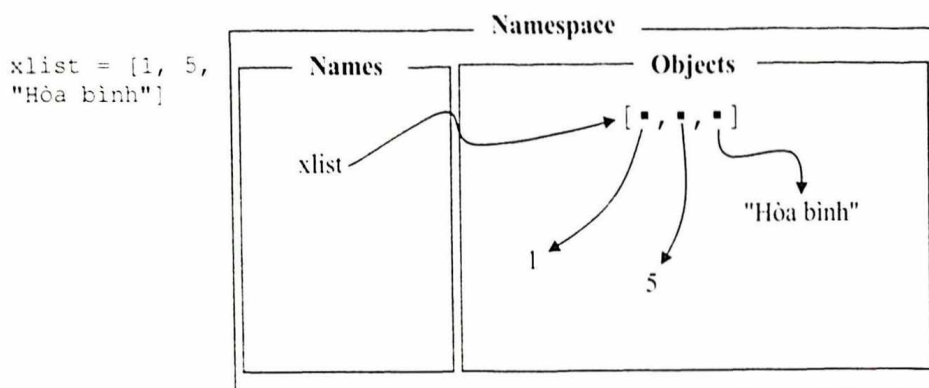
Tại đây, `xlist` được gán giá trị ban đầu.

Khi thực hiện xong hàm số, giá trị `xlist` không thay đổi.

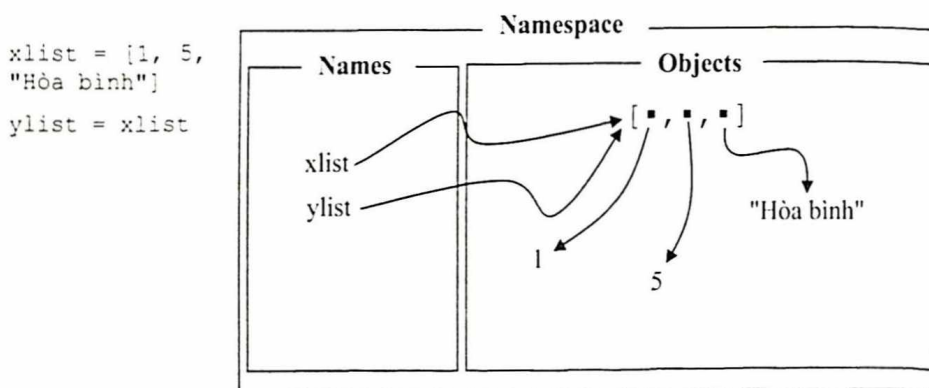
Với 2 ví dụ sau cùng, dáng điệu của biến list lại giống như biến nhớ dạng chuẩn mà chúng ta đã biết. Vì sao lại như vậy? Giải thích sẽ có trong phần tiếp theo.

6.2. Giải thích liên kết tham chiếu list

Chúng ta đã biết Python kiểm soát các biến nhớ thông qua hệ thống Namespace. Hệ thống này sẽ quản lý 2 loại: Hệ thống tên và các đối tượng. Khi thực hiện lệnh gán, ví dụ `x = 5` Python sẽ tạo liên kết từ tên `x` đến đối tượng số là 5. Với list thì cách quản lý cũng tương tự, tuy nhiên đối tượng list sẽ được hình dung như một cái thùng (container) chứa bên trong các đối tượng khác. Hình sau mô tả trạng thái khi gán `xlist = [1, 5, "Hòa bình"]`. Mỗi phần tử của list sẽ liên kết đến các đối tượng dữ liệu chuẩn khác.



Nếu thực hiện lệnh `ylist = xlist` thì 2 tên `xlist`, `ylist` cùng chỉ đến cùng một container. Chú ý rằng có 2 tên biến nhớ nhưng cùng chỉ vào chỉ một đối tượng list.

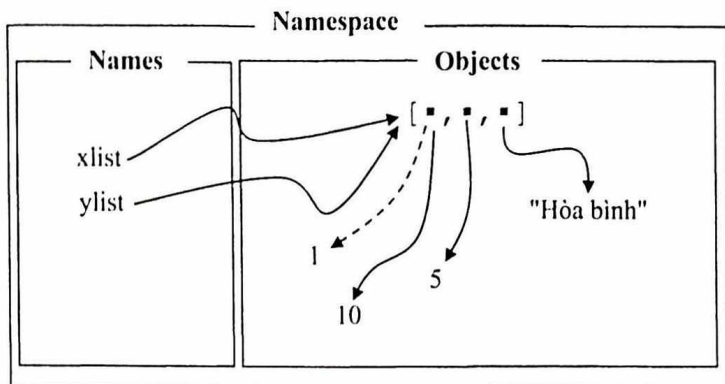


Bây giờ chúng ta thực hiện lệnh `xlist[0] = 10`.

Từ hình sau chúng ta thấy đối tượng container không thay đổi, mà chỉ một phần tử đầu tiên của list này chỉ sang một đối tượng khác là 10.

Do đó sau khi thực hiện lệnh trên chúng ta vẫn có `xlist = ylist`.

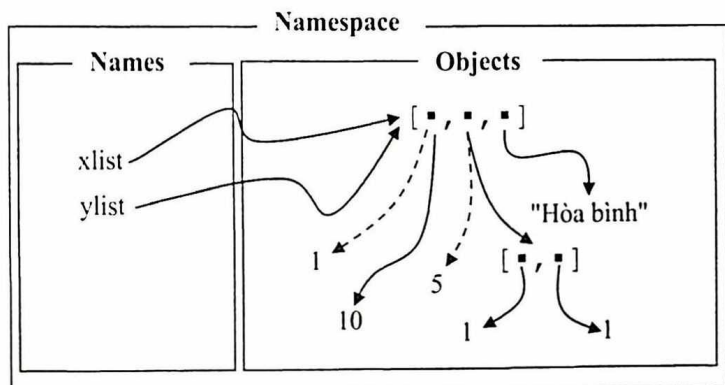
```
xlist = [1, 5,
"Hòa bình"]
ylist = xlist
xlist[0] = 10
```



Bây giờ chúng ta lại thực hiện lệnh `ylist[1] = [1, 1]`.

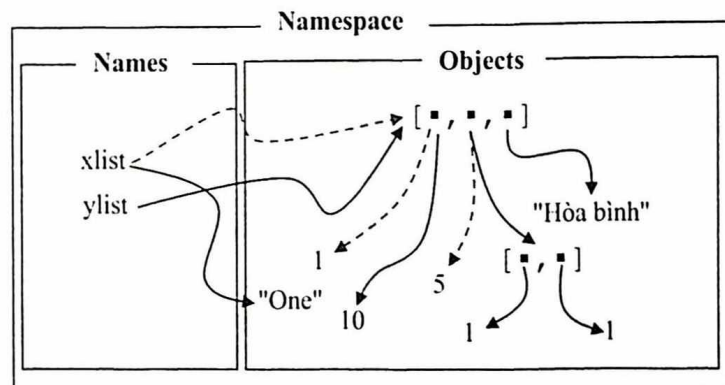
Phần tử thứ 2 của đối tượng container sẽ chỉ sang một đối tượng container khác cũng là list, đối tượng này là `[1, 1]`. Do đó `xlist` và `ylist` vẫn chỉ vào cùng một container, do đó chúng ta vẫn có `xlist = ylist`.

```
xlist = [1, 5,
"Hòa bình"]
ylist = xlist
xlist[0] = 10
ylist[1] = [1, 1]
```



Nếu bây giờ thực hiện lệnh `xlist = "One"`, `xlist` sẽ chỉ sang một đối tượng khác, do đó bây giờ `xlist` và `ylist` sẽ khác nhau.

```
xlist = [1, 5,
"Hòa bình"]
ylist = xlist
xlist[0] = 10
ylist[1] = [1, 1]
xlist = "One"
```



7. Chức năng nâng cao: list comprehension

Giả sử chúng ta muốn thực hiện một tính toán nào đó trên tất cả các phần tử của một danh sách list, khi đó chúng ta phải thực hiện lần lượt các tính toán đó trên từng phần tử của list và đưa vào một danh sách mới.

Xét ví dụ sau: cho trước danh sách x, cần thiết lập một danh sách bao gồm các phần tử lấy từ x sau khi đã bình phương lên. Chúng ta có thể dùng đoạn chương trình sau:

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> y = []
>>> for item in x:
>>>     y.append(item*item)
>>> y
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Hoặc có thể định nghĩa một hàm số để tạo ra kết quả là list mới như sau.

```
>>> def bp(x):
>>>     y = []
>>>     for item in x:
>>>         y.append(item*item)
>>>     return y
>>> y = bp(x)
>>> y
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Bây giờ chúng ta sẽ sử dụng một tính năng mới, nâng cao với list của Python. Chức năng này gọi là list comprehension.

Cú pháp sau sẽ tạo ra một list mới có số phần tử bằng <list> nhưng mỗi phần tử được tính thông qua <biểu thức> của các phần tử của <list>.

```
[<biểu thức> for item in <list>]
```

Giả sử cho trước dãy số trong danh sách list xlist. Khi đó lệnh:

```
[item*item for item in xlist]
```

sẽ tạo ra một list mới có các phần tử là bình phương các phần tử tương ứng của xlist.

Đoạn lệnh như sau.

```
>>> xlist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> ylist = [item*item for item in xlist]
>>> ylist
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

8. Đối tượng dữ liệu tuần tự (iterable)

8.1. Dữ liệu kiểu tuần tự

Chúng ta đã được làm quen với rất nhiều dạng dữ liệu tuần tự, tức là đối tượng dữ liệu mà có thể truy cập lần lượt đến từng phần tử bằng lệnh for. Đó là:

- List.
- Tuple.

- Xâu ký tự.
- Range.
- List Comprehension.

Xét ví dụ sau:

```
>>> R = range(10)
>>> R
range(0, 10)
>>> for i in R:
    print(i, end = " ")
0 1 2 3 4 5 6 7 8 9
```

Khi gán `R = range(10)` chúng ta đã tạo ra một biến nhớ mới `R` đồng nhất với `range(10)` và là một kiểu dữ liệu tuần tự khác.

Chú ý quan trọng:

1. Trong các chương tiếp theo chúng ta sẽ thấy một số kiểu dữ liệu nâng cao khác như từ điển, tập hợp cũng có khả năng liệt kê tuần tự như trên.
2. Các kiểu đối tượng sẽ là iterable nếu trong định nghĩa gốc của nó có các phương thức, hàm hệ thống `__iter__()` dùng cho lệnh `for` và `__getitem__()` dùng cho việc lấy giá trị theo chỉ số.

8.2. Tạo nhanh dữ liệu kiểu tuần tự

Để tạo nhanh một kiểu đối tượng tuần tự chúng ta có thể dùng cách làm tương tự như list comprehension nhưng biểu thức để trong dấu ngoặc tròn. Một kiểu dữ liệu được tạo như vậy gọi là một **generator**. Ví dụ:

```
>>> G = (2*i + 1 for i in range(10))
>>> type(G)
<class 'generator'>
```

Đối tượng `G` được khởi tạo thuộc kiểu **generator**.

Cú pháp tạo một biến kiểu tuần tự tổng quát như sau.

```
<generator> = (<biểu thức> for <tham biến> in <iterator>)
```

Chú ý quan trọng: Đối tượng kiểu tuần tự như trên chỉ có thể sinh dữ liệu được một lần. Ví dụ sau cho thấy sau khi khởi tạo nếu thực hiện lệnh duyệt bằng lệnh `for` thì biến kiểu tuần tự sẽ chỉ cho kết quả ở lần thực hiện đầu tiên.

```
>>> for k in G:
    print(k, end = " ")
1 3 5 7 9 11 13 15 17 19
>>> for k in G:
    print(k, end = " ")
```

Với lệnh này, `G` sẽ sinh được các dữ liệu tuần tự.

Với lệnh thứ hai này, `G` sẽ không tạo dữ liệu được nữa.

Để tạo được đối tượng kiểu dữ liệu tuần tự dùng nhiều lần chúng ta cần tạo ra các kiểu dữ liệu chuẩn như list, tuple, range, string hoặc dùng cách tạo **generator** đặc biệt được mô tả trong phần sau.

8.3. Tạo hàm generator bằng lệnh yield

Cách tạo các **generator** dùng nhiều lần rất giống với tạo hàm số, điều khác biệt ở chỗ từ khóa **return** được thay thế bằng từ khóa **yield** và có thể dùng **yield** nhiều lần, mỗi lần dùng **yield** là một lần tạo khả năng sinh dữ liệu. Ta gọi chúng là các **hàm sinh tuần tự** (generator function).

Ví dụ sau sẽ tạo một hàm generator **OddNum(n)** sinh dãy n số lẻ đầu tiên.

```
def OddNum(n):  
    for i in range(n):  
        yield 2*i + 1
```

Hàm **OddNum()** có thể được dùng như một kiểu dữ liệu tuần tự như sau, và có thể dùng nhiều lần.

```
>>> n = 20  
>>> for k in OddNum(n):  
    print(k, end = " ")  
  
>>> 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

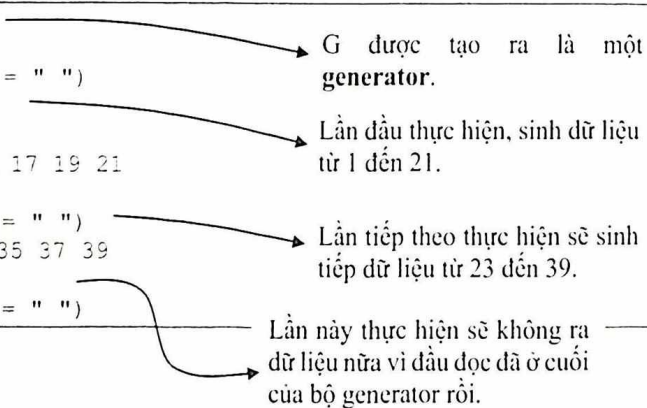
8.4. Sự khác nhau giữa generator và các kiểu dữ liệu tuần tự khác

Vì sao lại cần các **generator** trong khi chúng ta có thể tạo ra các bộ dữ liệu tuần tự theo các kiểu chuẩn khác như list, tuple, string?

Câu trả lời nằm trong các ý sau:

- Kiểu **generator** không chiếm bộ nhớ như dữ liệu tuần tự khác (list, tuple, string). Khi thực hiện việc sinh dữ liệu từ các **generator**, các máy tạo dữ liệu này mới khởi động và sinh dữ liệu tuần tự.
- Tốc độ sinh dữ liệu bằng **generator** cũng nhanh gần như tức thời, tương đương với việc lấy dữ liệu từ các đối tượng dữ liệu chuẩn như list, tuple hay string. Do vậy dùng **generator** sẽ không làm hạn chế tốc độ chương trình.
- Bản thân các **generator** có tính năng sinh và kết xuất dữ liệu đúng một lần. Khi đọc dữ liệu từ **generator**, đầu đọc của dữ liệu sẽ di chuyển tuần tự từ đầu đến cuối và chỉ đọc một lần. Đây là sự khác biệt rất lớn giữa **generator** và các kiểu dữ liệu tuần tự khác. Ví dụ được minh họa trong đoạn chương trình sau:

```
>>> G = OddNum(20)  
>>> for k in G:  
    print(k, end = " ")  
    if k > 20:  
        break  
1 3 5 7 9 11 13 15 17 19 21  
>>> for k in G:  
    print(k, end = " ")  
23 25 27 29 31 33 35 37 39  
>>> for k in G:  
    print(k, end = " ")
```



- G được tạo ra là một **generator**.
- Lần đầu thực hiện, sinh dữ liệu từ 1 đến 21.
- Lần tiếp theo thực hiện sẽ sinh tiếp dữ liệu từ 23 đến 39.
- Lần này thực hiện sẽ không ra dữ liệu nữa vì đầu đọc đã ở cuối của bộ generator rồi.



Câu hỏi - bài tập

1. Đoạn chương trình sau in ra kết quả nào?

```
Xlist = [1, [1, 2], [1, 2, 3], [1, 2, 3, 4]]  
print(Xlist[2] + [1])
```

- a) 4
- b) [1, 2, 3, 1]
- c) [[1, 2], 1]
- d) [[1, 2, 3], [1, 2, 3, 1]]

2. Kết quả đoạn chương trình sau là bao nhiêu?

```
X = [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4]]  
sum = 0  
for j in range(len(X)):  
    for i in range(len(X[j])):  
        sum = sum + X[j][i]  
print(sum)
```

- a) 0
- b) 10
- c) 20
- d) 50

3. Kết quả đoạn chương trình sau như thế nào?

```
for i in range(4):  
    for j in range(3):  
        print(i * j, end = " ")
```

- a) 0123456789101112
- b) 00001111222233334444
- c) 0000124246
- d) 000012024036

4. Đoạn chương trình sau kết quả thể hiện như thế nào?

```
>>> x = [1]  
>>> y = x  
>>> x = x + [2]  
>>> print(x, y)
```

Hãy giải thích kết quả trên.

5. Đoạn chương trình sau kết quả thể hiện như thế nào?

```
>>> x = [1]  
>>> y = x  
>>> x[0] = 15  
>>> print(x, y)
```

Hãy giải thích kết quả trên.

6. Kết quả của các lệnh sau là gì?

```
>>> s = "cable"  
>>> s[3 : -6: -1]
```

- a) 'elbac'
- b) 'lbac'
- c) 'lbace'
- d) 'bacel'

7. Sau khi chạy chương trình sau, các dãy B, C sẽ như thế nào?

```
B = C = []  
for i in range(3):  
    B.append(i)  
for i in range(4):  
    C.append(i)
```

- a) B = [0, 1, 2], C = [0, 1, 2, 3].
- b) B = [0, 1, 2, 0, 1, 2, 3], C = [0, 1, 2, 3].
- c) B = [0, 1, 2], C = [0, 1, 2, 0, 1, 2, 3].
- d) B = [0, 1, 2, 0, 1, 2, 3], C = [0, 1, 2, 0, 1, 2, 3].

8. Ma trận $m \times n$ (m hàng, n cột) có thể được mô tả bởi danh sách như sau:

$A = [[a_{11}, a_{12}, \dots, a_{1n}], [a_{21}, a_{22}, \dots, a_{2n}], \dots, [a_{m1}, a_{m2}, \dots, a_{mn}]]$

Cho trước ma trận (hay list) A, viết chương trình tính:

- Số hàng của ma trận trên.
- Số cột của ma trận trên.

9. Giả sử A là dãy mô tả ma trận $m \times n$ như bài trên. Ví dụ:

$A = [[1, 0, 5], [13, -1, 0], [2, 16, -18], [10, 9, -1]]$

Hỏi sau khi chạy chương trình sau, các dãy B, C thể hiện thông tin gì?

```
1. B = []  
2. C = []  
3. for item in A:  
4.     sum = 0  
5.     for i in item:  
6.         sum = sum + i  
7.     B.append(sum)  
8. n = len(A[0])  
9. for j in range(n):  
10.    sum = 0  
11.    for item in A:  
12.        sum = sum + item[j]  
13.    C.append(sum)
```

10. Giả sử A là list mô tả ma trận $m \times n$ như các bài tập trên.

$A = [[a_{11}, a_{12}, \dots, a_{1n}], [a_{21}, a_{22}, \dots, a_{2n}], \dots, [a_{m1}, a_{m2}, \dots, a_{mn}]]$

Viết thủ tục **ChangeRow(A,i,j)** có chức năng đổi vị trí 2 hàng i, j của ma trận A.

11. Giả sử A là list mô tả ma trận $m \times n$ như các bài tập trên.

$$A = [[a_{11}, a_{12}, \dots, a_{1n}], [a_{21}, a_{22}, \dots, a_{2n}], \dots, [a_{m1}, a_{m2}, \dots, a_{mn}]]$$

Viết thủ tục **ChangeColumn(A, i, j)** có chức năng đổi vị trí 2 cột i, j của ma trận A.

12. Cho 2 list biểu diễn ma trận A, B cùng kích thước. Viết thủ tục **Sum(A, B)** trả lại list C là tổng của 2 ma trận A, B.

13. Cho 2 list biểu diễn ma trận A, B cùng kích thước. Viết thủ tục **Diff(A, B)** trả lại list C là hiệu của 2 ma trận A, B.

14. Cho dãy A biểu diễn ma trận kích thước $m \times n$.

Viết thủ tục **Scalar(A, k)** trả lại list biểu diễn ma trận $C = kA$, phép nhân ma trận A với số thực k.

15. Cho 2 ma trận A, B kích thước $m \times n$ và $n \times p$.

Viết thủ tục **Multiple(A, B)** trả lại list là ma trận C là tích của 2 ma trận A, B.

Theo định nghĩa ma trận tích $C = A \times B$ có kích thước $m \times p$ và được định nghĩa như sau:

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}$$

16. Chứng minh rằng với dãy (list, tuple, string) có độ dài len thì quan hệ giữa chỉ số dương và chỉ số âm được cho trong bảng sau:

Chỉ số dương	Chỉ số âm
p	p - len
q + len	q

17. Cho trước dãy số bất kỳ:

$$a_1, a_2, \dots, a_n \quad (*)$$

Các phần tử của dãy có thể được nhập từ bàn phím hoặc cho trước trong chương trình.

Viết các chương trình thực hiện các công việc sau:

- Kiểm tra xem dãy (*) có phải là đơn điệu tăng hay không?
- Kiểm tra xem dãy (*) có phải là đơn điệu tăng thực sự hay không?
- Kiểm tra xem dãy (*) có phải là đơn điệu giảm hay không?
- Kiểm tra xem dãy (*) có phải là đơn điệu giảm thực sự hay không?

18. Cho trước dãy số bất kỳ:

$$a_1, a_2, \dots, a_n \quad (*)$$

Các phần tử của dãy có thể được nhập từ bàn phím hoặc cho trước trong chương trình.

Viết chương trình tìm một dãy con liên tục đơn điệu tăng lớn nhất trong dãy trên.

Chú ý: Chỉ cần chỉ ra một dãy con như vậy.

Ví dụ nếu dãy ban đầu là:

3 1 2 4 -1 8 7 10 11 7 8 9 0 16 40 50 -100

Dãy con liên tục đơn điệu tăng dài nhất là 0 16 40 50.

19. Cho trước một dãy số chỉ bao gồm các số 0 và 1.

Viết chương trình chỉ ra vị trí và độ dài của một dãy con liên tục dài nhất toàn các số 0.

20. Giả sử A là list mô tả ma trận $m \times n$ như trong bài tập 8.

$$A = [[a_{11}, a_{12}, \dots, a_{1n}], [a_{21}, a_{22}, \dots, a_{2n}], \dots, [a_{m1}, a_{m2}, \dots, a_{mn}]]$$

Ma trận A được mô tả như list trên chính là ma trận sau:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Phép lấy đối xứng của ma trận A là ma trận sau:

$$A^T = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Viết chương trình cho trước ma trận A và tính ma trận A^T .

21. Mô tả ý nghĩa và kết quả thực hiện các lệnh sau từ Python Shell.

```
>>> G = (2*i + 1 for i in range(10))
>>> for i in G:
    print(i)
    if i == 9:
        break
>>> for k in G:
    print(k, end = " ")
```

22. Viết hàm sinh dữ liệu tuần tự theo yêu cầu sau:

a) Hàm sinh n số chẵn đầu tiên tính từ 0.

b) Hàm sinh n số chính phương đầu tiên.

23. Viết hàm GenPrime(n) sinh generator bao gồm n số nguyên tố đầu tiên.

24. Nếu trong bài tập 21 thay $G = \text{range}(1, 20, 2)$ thì kết quả có thay đổi gì không?

25. Sử dụng chức năng list comprehension để tạo các dãy (list) sau:

a) Dãy tất cả các số là bội của 3 và nhỏ hơn 100.

b) Dãy 20 số chính phương đầu tiên.

c) Dãy 10 số nhị phân theo quy luật: 1, 10, 100, ...

d) Dãy 10 số nhị phân theo quy luật: 1000000000, 0100000000, ..., 0000000001.

26. Viết chương trình sinh dãy (list) A là biểu diễn của ma trận đơn vị.

27. Viết hàm số newMatrix(n) sinh ma trận vuông $n \times n$ bao gồm toàn các số 0.

28. Viết hàm số newMatrix(m, n) sinh ma trận kích thước $m \times n$ bao gồm toàn các số 0.

Chương 8. Khái niệm Module

Mục tiêu:

- Khái niệm module.
- Module math và cmath.
- Tự thiết lập module.
- Module random

Nội dung chính

Python được xây dựng với một lõi nhỏ gọn với lý do cho phép tất cả mọi người đều có thể bổ sung thêm các thư viện hàm số để sử dụng. Các thư viện hàm số đó được gọi là module. Bên cạnh lõi chính, khi cài đặt Python còn sao chép nhiều module để người sử dụng có thể dùng. Đây là một tính năng rất đặc biệt của Python.

Khi mở Python Shell, chúng ta thử kiểm tra xem trong lõi gốc của Python có những hàm, thủ tục nào. Thực hiện và quan sát đoạn chương trình sau.

```
>>> x = 1
>>> a = [1, 2]
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__', 'a', 'x']
>>>
```

Chúng ta chú ý đến từ khóa `__builtins__`, phương thức này sẽ liệt kê cho chúng ta tất cả các hàm hệ thống nằm trong lõi gốc của Python.

Hãy cùng quan sát các hàm lõi này.

```
>>> dir(__builtins__)
[.....
'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint',
'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict',
'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter',
'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance',
'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map',
'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open',
'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted',
'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars',
'zip']
```

Chúng ta có nhận xét ngay rằng số lượng các hàm hệ thống lõi của Python không nhiều. Trên thực tế hầu hết các ứng dụng đều phải được xây dựng bằng cách bổ sung các module có sẵn của Python hoặc từ các module được làm mới.

Trong danh sách các hàm hệ thống trên có khá nhiều hàm số chúng ta đã được làm quen trong các bài học trên như `len()`, `input()`, `print()`, `dir()`.

Một vài hàm được in đậm trong danh sách trên thể hiện trong ví dụ sau.

```
>>> xlist = [1, 0, 101, -20, 15, 78]
>>> min(xlist) # tìm phần tử nhỏ nhất của dãy số.
-20
>>> max(xlist) # tìm phần tử lớn nhất của dãy số.
101
>>> sum(xlist) # tính tổng các số của dãy.
175
>>> abs(1 + 1j) # hàm trị tuyệt đối. 1 + 1j là số phức 1 + i,
1.4142135623730951
>>>
```

Hàm **round(num, digit)** sẽ trả lại giá trị đã làm tròn của **num** với số chữ số sau dấu phẩy = **digit**, ví dụ:

```
>>> round(3.1416, 2)
3.14
```

1. Lệnh import

Muốn sử dụng một module, hay đúng hơn sử dụng các hàm của module đó chúng ta phải dùng lệnh **import**.

import <tên module>

Ví dụ: **math** là một module lớn của Python chứa rất nhiều hàm và hằng số toán học. Muốn sử dụng các hàm này cần thực hiện lệnh.

import math

Ví dụ hàm **sqrt()** của module **math** là hàm tính căn bậc 2. Muốn sử dụng hàm này thực hiện như sau:

```
>>> math.sqrt(2)
1.4142135623730951
>>>
```

Như vậy cách truy cập một phương thức của module đã **import** thì dùng lệnh sau:

<module>.<tên phương thức>

Chú ý nếu sau khi đã **import math** nhưng dùng hàm trực tiếp, ví dụ **sqrt(2)** thì vẫn bị lỗi, mà phải viết **math.sqrt(2)**.

```
>>> sqrt(2)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    sqrt(2)
NameError: name 'sqrt' is not defined
>>>
```

Nếu muốn sử dụng trực tiếp hàm số của module thì có thể dùng lệnh sau:

from <module> **import** <hàm số>

hoặc dùng lệnh sau sẽ cho phép sử dụng trực tiếp tất cả các hàm của module.

from <module> **import ***

Ví dụ lệnh sau sẽ chuyển dùng hàm `sqrt()` trực tiếp mà không cần thông qua đối tượng `math`.

```
>>> from math import sqrt
>>> sqrt(2)
1.4142135623730951
>>>
```

Có thể dùng lệnh `import` để thực hiện với nhiều module, cú pháp như sau.

```
import <module1>, <module2>, ..., <moduleN>
```

Có thể dùng lệnh `import` như sau:

```
>>> import math as m
>>> m.sqrt(2)
1.4142135623730951
>>>
```

2. Module math

Chúng ta cùng quan sát kỹ hơn module `math`.

Lệnh sau sẽ xem tất cả các phương thức, tham biến có trong module `math`.

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc',
'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'logip',
'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Một số phương thức in đậm sẽ được thử trong ví dụ sau.

Ví dụ:

```
>>> math.pi           # hằng số Pi
3.141592653589793
>>> math.e             # hằng số e
2.718281828459045
>>>
```

3. Số phức và module cmath

Python hỗ trợ dữ liệu số là số phức. Số phức trong Python qui định viết như sau:

<giá trị phần thực> + <giá trị phần ảo>j

Ví dụ:

```
>>> z = 3 + 2j
>>> z
(3+2j)
>>> type(z)
<class 'complex'>
>>>
```

Chú ý:

- Nếu giá trị ảo của số phức là 1 thì phải viết 1j.
- Không được viết j trước giá trị số.

Ví dụ:

```
>>> 1 + j          # không được viết j, bắt buộc phải viết 1j
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    1 + j
NameError: name 'j' is not defined
>>> 1 + j5         # không được viết số sau j, phải viết 5j
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    1 + j5
NameError: name 'j5' is not defined
>>>
```

Chúng ta cùng xem một số phương thức của lớp số phức trong Python.

```
>>> dir(z)
['_abs_', '_add_', '_bool_', '_class_', '_delattr_',
'_dir_', '_divmod_', '_doc_', '_eq_', '_float_',
'_floordiv_', '_format_', '_ge_', '_getattr_',
'_getnewargs_', '_gt_', '_hash_', '_init_',
'_init_subclass_', '_int_', '_le_', '_lt_', '_mod_',
'_mul_', '_ne_', '_neg_', '_new_', '_pos_', '_pow_',
'_radd_', '_rdivmod_', '_reduce_', '_reduce_ex_',
'_repr_', '_rfloordiv_', '_rmod_', '_rmul_', '_rpow_',
'_rsub_', '_rtruediv_', '_setattr_', '_sizeof_',
'_str_', '_sub_', '_subclasshook_', '_truediv_',
'conjugate', 'imag', 'real']
```

Các phương thức in đậm ở trên là các hàm trả lại số phức liên hợp (**conjugate**), tính phần ảo (**imag**) và phần thực (**real**) của một số phức.

```
>>> z = 5 + 4j
>>> z.conjugate()
(5-4j)
>>> z.real
5.0
>>> z.imag
4.0
>>>
```

Chú ý:

real và **imag** là thuộc tính, không phải là phương thức.

Module `cmath` của Python bao gồm các hàm, hằng và phương thức hỗ trợ tính toán trên các số phức.

Sau đây là danh sách các phương thức của module này.

```
>>> dir(cmath)
['_doc_', '_loader_', '_name_', '_package_', '_spec_',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atanh', 'cos', 'cosh',
'e', 'exp', 'inf', 'infj', 'isclose', 'isfinite', 'isinf',
'isnan', 'log', 'log10', 'nan', 'nanj', 'phase', 'pi', 'polar',
'rect', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau']
```

Ví dụ một số hàm, hằng được in đậm trong danh sách trên.

```
>>> cmath.e           # hằng số e
2.718281828459045
>>> cmath.pi          # hằng số pi
3.141592653589793
>>> cmath.sqrt(1 + 1j) # tính căn bậc 2 của số phức 1 + i
(1.09868411346781+0.45508986056222733j)
>>>
```

4. Tự thiết lập module

Python cho phép người lập trình dễ dàng tạo ra các module cho riêng mình và chia sẻ trên cộng đồng. Việc tạo các module này rất đơn giản: Mỗi module như vậy là một tệp chương trình *.py, trong đó tất cả các hàm số và biến nhớ được định nghĩa sẽ là tài nguyên của module này.

Ví dụ tệp module sau có 2 hàm số f1() và f2() được định nghĩa.

module_1.py

```
1. # sample of module
2. def f1(x):
3.     print("Đây là số cần in:", x);
4. def f2(x, y):
5.     return x*x + y*y
```

Muốn đưa module này vào sử dụng cần copy tệp này vào thư mục Lib các thư viện gốc của Python.

Chương trình sau sẽ mô tả cách sử dụng module trên và các hàm của module này.

Chú ý đến dòng 4, lệnh gọi trực tiếp hàm f2() bị lỗi, cần phải gọi đúng như tại dòng 9.

```
1. >>> import module_1
2. >>> dir(module_1)
3. ['__builtins__', '__cached__', '__doc__', '__file__',
   '__loader__', '__name__', '__package__', '__spec__', 'f1',
   'f2']
4. >>> f2(3, 4)           # lỗi gọi không đúng
5. Traceback (most recent call last):
6.   File "<pyshell#4>", line 1, in <module>
7.     f2(3, 4)
8. NameError: name 'f2' is not defined
9. >>> module_1.f2(3, 4)  # lỗi gọi đúng.
10. 25
```

Để có thể gọi trực tiếp các hàm số từ module chúng ta thực hiện theo cách ghi ra ở chương trình sau.

```
>>> from module_1 import *
>>> f1(16)
Đây là số cần in: 16
>>> f2(9, 10)
181
```


Sau đây là một ví dụ hoàn chỉnh chương trình gọi và sử dụng module. Trong ví dụ sau tệp module cần để cùng một thư mục với tệp chạy.

sample.py

```
1. # sample of using module
2. from module_1 import *
3. x = 3
4. y = 4
5. f1(x + y)
6. print(f2(x, y))
```

5. Sinh số ngẫu nhiên trong Python

Python có một module tên là **random**. trong module này có hàm số **random()** có chức năng sinh ngẫu nhiên một số thực nằm trong khoảng $[0, 1)$. Từ hàm này chúng ta có thể thực hiện nhiều bài toán sinh số ngẫu nhiên.

Cách gọi hàm **random()** có thể như sau.

```
>>> from random import random
>>> random()
0.19343934679311237
>>>
```

Chú ý hàm **random()** không có tham số.

Sau đây là một vài ứng dụng của hàm này.

1. Sinh các số nguyên ngẫu nhiên nằm trong khoảng $[0, 100]$

```
from random import random
>>> int(101*random())      # sinh số nguyên ngẫu nhiên  $[0, 100]$ 
6
>>> for i in range(100):   # sinh 100 số nguyên ngẫu nhiên  $< 100$ 
    print(int(101*random()), end = " ")
84 58 9 61 2 14 36 83 13 25 4 4 48 27 72 23 5 19 48 16 4 87 34 29
12 0 83 29 50 87 68 67 40 64 74 22 53 77 15 3 38 26 62 49 64 12
73 46 48 67 86 21 21 84 77 78 54 21 85 50 23 68 46 90 61 48 73 47
68 54 94 20 34 23 1 65 18 7 21 73 4 71 56 78 7 0 5 92 53 20 72 39
55 66 79 84 6 27 81 32
>>>
```

2. Sinh các số thực ngẫu nhiên nằm trong khoảng $[0, 100]$

Số ngẫu nhiên trong khoảng $[0, 100]$ được tính bằng $100*\text{random}()$.

```
from random import random
>>> x = 100*random()
>>> x
3.0330662762006577
>>>
```

3. Sinh các số tự nhiên ngẫu nhiên trong khoảng $[M, N]$ ($M < N$ là các số tự nhiên)

Công thức tính số ngẫu nhiên được sinh ra trong đoạn $[M, N]$ được tính như sau:

$M + (N + 1 - M) * \text{random}()$.

```
>>> N = 100
>>> M = 50
>>> for i in range(100):
    print(int(M + (N+1-M)*random()), end = " ")
88 86 67 90 72 57 63 56 56 75 69 85 94 57 94 54 73 84 83 87 68 74
90 99 85 82 96 61 62 78 87 80 89 56 50 95 52 60 95 59 80 66 99 90
67 75 65 68 98 58 56 68 66 78 66 96 87 93 66 84 87 90 79 84 67 58
96 67 87 59 75 91 64 81 61 58 99 76 52 57 94 71 64 57 82 75 78 58
86 56 93 74 86 96 77 63 90 65 93 88
>>>
```

4. Viết chương trình nhập 2 số tự nhiên N, P và sinh một dãy N số tự nhiên ngẫu nhiên trong khoảng [0, P).

Chương trình đơn giản sau sẽ yêu cầu nhập từ bàn phím 2 số tự nhiên N, P, sau đó tạo dãy A gồm N số tự nhiên ngẫu nhiên trong khoảng [0, P).

Sinh_dãy_ngẫu_nhiên.py

```
1. from random import random
2. N = int(input("Nhập số phần tử cần sinh của dãy: "))
3. P = int(input("Nhập giá trị MAX của dãy cần sinh: "))
4. A = []
5. for i in range(N):
6.     A.append(int(P*random()))
7. print(A)
```

Kết quả chạy chương trình có thể như sau.

```
Nhập số phần tử cần sinh của dãy: 20
Nhập giá trị MAX của dãy cần sinh: 100
[70, 85, 31, 44, 39, 62, 16, 5, 78, 5, 89, 19, 46, 2, 21, 33, 56,
29, 1, 71]
```

6. Sinh giá trị ngẫu nhiên từ một dãy

Trong module random ngoài hàm random() ra còn một hàm khác rất hay sử dụng, đó là hàm choice().

Cú pháp hàm này như sau.

```
choice(<dãy giá trị tuần tự>)
```

Tham số của hàm này có thể là một đối tượng dãy tuần tự bất kỳ, ví dụ list, tuple hay xâu ký tự. Hàm sẽ trả lại một phần tử ngẫu nhiên của dãy tuần tự trong tham số.

Sau đây là các ví dụ:

- Sinh ngẫu nhiên một số trong khoảng từ 0 đến 100.

```
>>> import random as r
>>> r.choice(range(100))
91
>>>
```

- Sinh ngẫu nhiên một phần tử lấy từ dãy số A cho trước.

```
>>> import random as r
>>> A = [0, 4, 19, 700, 15, 3201, -15, 35]
>>> r.choice(A)
15
>>>
```

- Sinh ngẫu nhiên một ký tự lấy từ một xâu cho trước.

```
>>> import random as r
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> r.choice(s)
'v'
>>>
```

- Sinh 100 số tự nhiên ngẫu nhiên < 100000.

```
>>> import random as r
>>> for i in range(100):
    print(r.choice(range(1,100000)), end = " ")
50732 94381 16312 45082 21005 48077 34059 91461 97743 74395 25475
20710 30959 67852 45479 16994 43231 337 87175 72998 54926 56165
14905 34823 49549 85224 5533 82587 51600 86123 30768 26436 54197
59170 23875 74772 72910 15077 54138 40129 19281 25485 37264 46190
27672 61270 87111 70894 2964 56122 85884 15492 1199 89336 67323
37416 85160 38091 83530 58792 52950 6487 32865 53403 23627 24117
40509 77454 51129 87213 75660 90650 99927 57802 51733 89017 13167
53727 76620 43276 36877 20810 84144 21742 35490 26442 42503 5635
63994 19869 97951 5148 82071 11895 50050 66596 57108 45665 41405
63631
>>>
```

Ngoài 2 hàm sinh ngẫu nhiên quan trọng trên, module random còn có một số hàm sinh ngẫu nhiên khác sau đây.

randint(a, b)	Sinh ngẫu nhiên một số nguyên nằm giữa 2 giá trị [a, b] tính cả 2 đầu mút.
randrange(start, stop [, step])	Sinh ngẫu nhiên số nguyên nằm trong vùng của lệnh range(start, stop [, step]) tương ứng.



Câu hỏi - bài tập

1. Lệnh:

```
>>> abs(1 + 1j)
```

trả lại giá trị bao nhiêu?

- a) 1
- b) 2
- c) 1.41421356
- d) Lệnh báo lỗi.

2. Lệnh:

```
>>>import cmath
>>> (1.5 + j).real
```

trả lại giá trị nào?

- a) 1
- b) 1.5
- c) 0
- d) Lệnh báo lỗi

3. Lệnh:

```
>>>import cmath
>>> (2 + 3j).imag
```

trả lại giá trị nào?

- a) 2
- b) 3
- c) 5
- d) Lệnh báo lỗi.

4. Viết chương trình sinh một dãy (list) A bao gồm 1000 số tự nhiên ngẫu nhiên nằm trong khoảng [1, 99999].

5. Viết chương trình sinh một dãy (list) A bao gồm 1000 số tự nhiên ngẫu nhiên nằm trong khoảng [1, 99999], sau đó sắp xếp lại dãy này theo thứ tự tăng dần.

6. Công thức tính diện tích tam giác có các cạnh a, b, c là:

$$S = \sqrt{p(p-a)(p-b)(p-c)}.$$

Ở đây p = nửa chu vi tam giác.

Viết hàm số tính diện tích tam giác với các tham số đầu vào là a, b, c - số đo các cạnh của tam giác.

7. Viết chương trình nhập tọa độ 3 đỉnh A, B, C và tính chu vi, diện tích tam giác ABC.

8. Cho trước tọa độ 4 đỉnh A, B, C, D, viết chương trình kiểm tra cạnh AB có cắt cạnh CD không?

9. Thiết lập một module **MyMatrix** bao gồm các phương thức bổ sung cho đối tượng List. Thiết lập các hàm sau:

- **isMatrix(A)**, kiểm tra xem list A có phải là biểu diễn ma trận hay không. Trả lại True nếu đúng và False nếu sai.

- **ChangeRow(A, i, j)**, hàm thay đổi 2 hàng i, j của Ma trận A. Nếu A không là ma trận hoặc không thể thực hiện được thì trả lại False, ngược lại nếu thực hiện thành công thì trả lại True.

- **ChangeColumn(A, i, j)**, thủ tục thay đổi 2 cột i, j của Ma trận A. Nếu A không là ma trận hoặc không thể thực hiện được thì trả lại False, ngược lại nếu thực hiện thành công thì trả lại True.

- **isSquare(A)**, hàm kiểm tra xem ma trận A có phải là ma trận vuông hay không.

- **GetSymmetry(A)**, thủ tục lấy đối xứng qua đường chéo chính của Ma trận vuông A. Hàm trả lại ma trận A^T là đối xứng của A.

10. Thiết lập module **MyTriange** chứa các hàm và thủ tục liên quan đến tam giác như sau:

- **isTriange(a, b, c)**, hàm kiểm tra xem 3 số a, b, c có thiết lập được tam giác hay không. Trả lại True nếu đúng và False nếu sai.

- **Perimeter(a, b, c)**, hàm tính chu vi của tam giác.

- **Area(a, b, c)**, hàm tính diện tích tam giác.

11. Chúng ta đã biết hằng số e có thể tính bằng công thức:

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

Thiết lập dãy (list) A bao gồm các phần tử sau:

$$1, (1 + 1/2)^2, (1 + 1/3)^3, \dots, (1 + 1/n)^n, \dots$$

Dãy này được thiết lập với độ dài N = 1000. Tính giá trị:

$$e - (1 + 1/N)^N.$$

12. Dùng lệnh Python nào để tính được các giá trị phức sau: e^i , e^{-i} .

13. Cho tam giác ABC với các cạnh có độ dài lần lượt là a, b, c. Các đường cao h_a , h_b , h_c tương ứng với các cạnh a, b, c được tính theo công thức:

$$h_a = 2S/a, h_b = 2S/b, h_c = 2S/c.$$

S là diện tích tam giác được tính theo công thức:

$$S = \sqrt{p(p-a)(p-b)(p-c)}.$$

Viết chương trình nhập 3 số a, b, c và tính các giá trị h_a , h_b , h_c . Kết quả thể hiện trên màn hình.

14. Cho trước số tự nhiên N. Sử dụng các hàm của module random để viết thủ tục sinh một hoán vị ngẫu nhiên của dãy 1, 2, ..., N. Kết quả thể hiện trên màn hình.

15. Cho trước dãy (list) A bao gồm N phần tử. Sử dụng các hàm của module random để viết thủ tục sinh một hoán vị ngẫu nhiên của dãy A. Kết quả thể hiện trên màn hình.

16. Viết hàm số kiểm tra xem số tự nhiên N có phải là số chính phương hay không. Gợi ý: Có thể sử dụng hàm **isqrt()** của module **math**.

17. Viết hàm số kiểm tra xem số tự nhiên N có phải là số nguyên tố hay không, yêu cầu sử dụng ít nhất phép chia lấy số dư. Gợi ý: Dùng hàm **floor()** của module **math**.

18. Viết hàm sinh generator **Primes(n)** trả lại dãy các số nguyên tố không vượt quá số tự nhiên n.

19. Viết hàm tính diện tích tam giác biết độ dài 2 cạnh a, b và góc α (tính số đo góc theo độ °), biết công thức tính diện tích là $S = ab \sin(\alpha)/2$.

20. Viết hàm tính chu vi và diện tích hình tròn biết bán kính R. Gợi ý: Sử dụng hằng số Pi của module **math**.

Chương 9. Xâu ký tự

Mục tiêu

- Kiểu dữ liệu str - xâu ký tự.
- Các phương thức làm việc chính với xâu ký tự.



Nội dung chính

1. Kiểu dữ liệu String

Chúng ta đã làm quen với kiểu dữ liệu xâu ký tự (str) trong bài học đầu tiên. Trong bài học này các bạn sẽ được học sâu và toàn diện hơn với kiểu dữ liệu đặc biệt này.

Tóm tắt một số đặc tính của kiểu dữ liệu string mà chúng ta đã biết.

- Dữ liệu string được khởi tạo bằng cách gán xâu ký tự nằm giữa hai dấu nháy đơn 'one two three' hoặc nháy kép "one two three".
- Tên kiểu dữ liệu: str.
- Xâu ký tự thuộc dạng dữ liệu tuần tự, do đó có thể thực hiện các hàm như len(), truy cập chỉ số, ví dụ s[i], hoặc vùng chỉ số, ví dụ s[i : j].
- Xâu ký tự không cho phép thay đổi từng ký tự của mình.
- Có thể thực hiện phép toán + xâu ký tự và phép toán * với một số tự nhiên.
- Trong Python không có kiểu dữ liệu ký tự, một ký tự cũng thuộc kiểu string.

Xem đoạn chương trình sau minh họa cho các tính chất trên.

```
>>> s = "Hà Nội Việt Nam"
>>> print(s)
Hà Nội Việt Nam
>>> type(s)
<class 'str'>
>>> len(s)
15
>>> for j in s:           # string là kiểu dữ liệu tuần tự
    print(j, end = " ")
H  à   N ội   V iệt   N ăm
>>> s[0]                 # truy cập từng ký tự của xâu.
'H'
>>> s[3]='V'             # không cho phép thay đổi dữ liệu.
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    s[3]='V'
TypeError: 'str' object does not support item assignment
>>> s*3                  # phép nhân với số nguyên
'Hà Nội Việt NamHà Nội Việt NamHà Nội Việt Nam'
>>> s + s                # phép cộng xâu ký tự
'Hà Nội Việt NamHà Nội Việt Nam'
>>>
```

Mọi đối tượng dữ liệu trong Python đều có thể chuyển đổi sang dạng string bằng hàm `str()`. Ví dụ:

```
>>> n = 100
>>> x = 3.1415
>>> a = [1, 2, "One", [3, 4]]
>>> s = str(n) + str(x) + str(a)
>>> s
"1003.1415[1, 2, 'One', [3, 4]]"
>>>
```

String là lớp dữ liệu cơ bản của Python với rất nhiều tính chất và phương thức kèm theo.

```
>>> dir(str)
['_add_', '__class__', '__contains__', '__delattr__',
'_dir_', '__doc__', '__eq__', '__format__', '__ge__',
'_getattr_', '__getitem__', '__getnewargs__', '__gt__',
'_hash_', '__init__', '__init_subclass__', '__iter__',
'_le_', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'_new_', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'_rmul__', '__setattr__', '__sizeof__', '__str__',
'_subclasshook_', 'capitalize', 'casefold', 'center', 'count',
'encode', 'endwith', 'expandtabs', 'find', 'format',
'format_map', 'index', 'isalnum', 'isalpha', 'isascii',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
>>>
```

Một số phương thức quan trọng in đậm trong danh sách trên sẽ được xem xét kỹ trong các phần tiếp theo của bài học này.

2. Bảng mã chuẩn ASCII và Unicode

Chúng ta cùng tìm hiểu cách máy tính lưu trữ ký tự và xâu ký tự. Máy tính chỉ làm việc với các con số, do đó các ký tự và xâu ký tự sẽ được mã hóa thành các con số để lưu trữ trong máy tính. Như vậy chắc chắn cần có một ánh xạ giữa tập các chữ cái và các số dùng để mã hóa các chữ cái này.

Ví dụ, nếu chữ `p` mã hóa thành 10, `e` mã hóa thành 12, `a` mã hóa thành 16, `c` mã hóa thành 15 thì chữ `peace` sẽ mã hóa thành 10 12 16 15 12.

```
p <-----> 10
e <-----> 12      mã hóa
a <-----> 16      peace <-----> 10 12 16 15 12
c <-----> 15      giải mã
```

Việc mã hóa các ký tự thành số là mô hình 2 chiều, mã hóa và giải mã.

Để thống nhất việc mã hóa trên, từ lâu cộng đồng khoa học đã thiết lập bảng mã chuẩn quốc tế dùng để qui định thống nhất một bảng mã duy nhất trên toàn thế giới. Bảng mã chuẩn quốc tế hiện nay vẫn dùng có tên là bảng ASCII (viết tắt của cụm từ **American Standard Code for Information Interchange**). Bảng mã gốc ban đầu ASCII có 256 ký tự, trong đó 128 ký tự đầu tiên là các ký tự chuẩn, 128 ký tự tiếp theo là các ký tự mở rộng. Trong bảng này mỗi ký tự sẽ được mã hóa thành một số nguyên duy nhất.

Ký tự <-----> số

Sau đây là 128 ký tự đầu tiên của bảng mã ASCII gốc.

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	\$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

Nhận xét:

- Mã hóa được đánh số từ 0.
- Các ký tự đánh số từ 0 đến 31 và ký tự mã hóa 127 là các ký tự điều khiển, không tương ứng và không thể hiện được trên màn hình.
- Các ký tự thực sự có mã hóa tương ứng từ 32 đến 126.
- Vùng các ký tự chữ số: 48 - 57.
- Vùng các ký tự chữ in hoa: 65 - 90, các ký tự chữ in thường: 97 - 122.

Hình sau cho ta quan sát bảng ASCII ở hình thức đẹp, trong đó ngoài mã hóa bằng số hệ thập phân còn ghi thêm số dạng hex (hệ đếm 16).

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0		32	20		64	40		96	C0	
1	1		33	21		65	41	A	97	61	a
2	2		34	22		66	42	B	98	62	b
3	3		35	23		67	43	C	99	63	c
4	4		36	24		68	44	D	100	64	d
5	5		37	25		69	45	E	101	65	e
6	6		38	26		70	46	F	102	66	f
7	7		39	27		71	47	G	103	67	g
8	8		40	28		72	48	H	104	68	h
9	9		41	29		73	49	I	105	69	i
10	A		42	2A		74	4A	J	106	6A	j
11	B		43	2B		75	4B	K	107	6B	k
12	C		44	2C		76	4C	L	108	6C	l
13	D		45	2D		77	4D	M	109	6D	m
14	E		46	2E		78	4E	N	110	6E	n
15	F		47	2F		79	4F	O	111	6F	o
16	10		48	30	0	80	50	P	112	70	p
17	11		49	31	1	81	51	Q	113	71	q
18	12		50	32	2	82	52	R	114	72	r
19	13		51	33	3	83	53	S	115	73	s
20	14		52	34	4	84	54	T	116	74	t
21	15		53	35	5	85	55	U	117	75	u
22	16		54	36	6	86	56	V	118	76	v
23	17		55	37	7	87	57	W	119	77	w
24	18		56	38	8	88	58	X	120	78	x
25	19		57	39	9	89	59	Y	121	79	y
26	1A		58	3A		90	5A	Z	122	7A	z
27	1B		59	3B		91	5B	[	123	7B	[
28	1C		60	3C	<	92	5C	\	124	7C	\
29	1D		61	3D	=	93	5D	]	125	7D	]
30	1E		62	3E	>	94	5E	^	126	7E	^
31	1F		63	3F	~	95	5F	_	127	7F	_

Hình sau là bảng ASCII đầy đủ với 256 phần tử.

ASCII control characters			ASCII printable characters			Extended ASCII characters		
00	NUL	(Null character)	32	space	64	@	96	
01	SOH	(Start of Header)	33	!	65	A	97	a
02	STX	(Start of Text)	34	"	66	B	98	b
03	ETX	(End of Text)	35	#	67	C	99	c
04	EOT	(End of Trans.)	36	\$	68	D	100	d
05	ENQ	(Enquiry)	37	%	69	E	101	e
06	ACK	Acknowledgement	38	&	70	F	102	f
07	BEL	(Bell)	39	'	71	G	103	g
08	BS	(Backspace)	40	(	72	H	104	h
09	HT	(Horizontal Tab)	41	)	73	I	105	i
10	LF	(Line feed)	42	*	74	J	106	j
11	VT	(Vertical Tab)	43	+	75	K	107	k
12	FF	(Form feed)	44	,	76	L	108	l
13	CR	(Carriage return)	45	-	77	M	109	m
14	SO	(Shift Out)	46	.	78	N	110	n
15	SI	(Shift In)	47	/	79	O	111	o
16	DLE	(Data Link escape)	48	0	80	P	112	p
17	DC1	(Device control 1)	49	1	81	Q	113	q
18	DC2	(Device control 2)	50	2	82	R	114	r
19	DC3	(Device control 3)	51	3	83	S	115	s
20	DC4	(Device control 4)	52	4	84	T	116	t
21	NAK	(Negative acknowledge)	53	5	85	U	117	u
22	SYN	(Synchronous idle)	54	6	86	V	118	v
23	ETB	(End of trans. block)	55	7	87	W	119	w
24	CAN	(Cancel)	56	8	88	X	120	x
25	EM	(End of medium)	57	9	89	Y	121	y
26	SUB	(Substitute)	58	:	90	Z	122	z
27	ESC	(Escape)	59	;	91	[	123	[
28	FS	(File separator)	60	<	92	\	124	\
29	GS	(Group separator)	61	=	93	]	125	]
30	RS	(Record separator)	62	>	94	^	126	^
31	US	(Unit separator)	63	?	95	_	127	_
128	DEL	(Delete)						
160	a		161	á		162	â	
163	û		164	ü		165	ý	
166	ø		167	ÿ		168	ÿ	
169	ÿ		170	ÿ		171	ÿ	
172	ÿ		173	ÿ		174	ÿ	
175	ÿ		176	ÿ		177	ÿ	
178	ÿ		179	ÿ		180	ÿ	
181	ÿ		182	ÿ		183	ÿ	
184	ÿ		185	ÿ		186	ÿ	
187	ÿ		188	ÿ		189	ÿ	
190	ÿ		191	ÿ		192	ÿ	
193	ÿ		194	ÿ		195	ÿ	
196	ÿ		197	ÿ		198	ÿ	
199	ÿ		200	ÿ		201	ÿ	
202	ÿ		203	ÿ		204	ÿ	
205	ÿ		206	ÿ		207	ÿ	
208	ÿ		209	ÿ		210	ÿ	
211	ÿ		212	ÿ		213	ÿ	
214	ÿ		215	ÿ		216	ÿ	
217	ÿ		218	ÿ		219	ÿ	
220	ÿ		221	ÿ		222	ÿ	
223	ÿ		224	ÿ		225	ÿ	
226	ÿ		227	ÿ		228	ÿ	
229	ÿ		230	ÿ		231	ÿ	
232	ÿ		233	ÿ		234	ÿ	
235	ÿ		236	ÿ		237	ÿ	
238	ÿ		239	ÿ		240	ÿ	
241	ÿ		242	ÿ		243	ÿ	
244	ÿ		245	ÿ		246	ÿ	
247	ÿ		248	ÿ		249	ÿ	
250	ÿ		251	ÿ		252	ÿ	
253	ÿ		254	ÿ		255	ÿ	
256	ÿ		257	ÿ		258	ÿ	

Bảng mã ASCII ban đầu có $256 = 2^8$ phần tử nên bảng này được gọi là bảng ký tự 8 bit. Hiện nay bảng mã này đã được mở rộng lên 16 bit, có thể chứa được $2^{16} = 65536$ ký tự. Bảng mã mới 16 bit này gọi là bảng mã Unicode. Python cũng đã hỗ trợ hoàn toàn cho bảng mã Unicode. Vậy bảng mã ASCII là một phần của bảng mã Unicode.

Chú ý: Bảng mã Unicode hỗ trợ hoàn toàn cho tiếng Việt. Tất cả các chữ cái tiếng Việt, kể cả có dấu thanh đều nằm trong bảng mã Unicode này.

Hàm `ord()` và `chr()`

Python có 2 hàm hệ thống là `ord()` và `chr()` dùng để biến đổi ký tự sang mã số và ngược lại.

`ord(<ký tự>)` = mã hóa của ký tự này trong bảng mã chuẩn (ASCII hoặc Unicode).

`chr(<giá trị>)` = ký tự tương ứng với mã hóa <giá trị> trong bảng mã chuẩn (ASCII hoặc Unicode).

Ví dụ.

```
>>> ord("A")
65
>>> ord("a")
97
>>> chr(65)
'A'
>>> ord("À")
7848
>>> chr(7848)
'À'
```

Ký tự "À" không có trong bảng ASCII nhưng có trong bảng mã Unicode với mã hóa 7848.

Quan sát bảng mã ASCII chúng ta sẽ nhận thấy một điều thú vị sau:

Mã hóa của chữ cái hoa luôn nhỏ hơn mã hóa chữ cái thường đúng 32 đơn vị. Hay nói cách khác giữa chữ cái thường và hoa luôn có tính chất sau:

`ord(<chữ cái hoa>) + 32 = ord(<chữ cái thường>)`

Chúng ta cùng xem một vài ứng dụng của hiện tượng trên.

1. Thiết lập hàm số để biến đổi một chuỗi ký tự từ chữ thường sang chữ hoa.

```
>>> def f_hoa(s):
    kq = ""
    for ch in s:
        kq = kq + chr(ord(ch)-32)
    return kq
>>> f_hoa("baby")
'BABY'
>>> f_hoa("table")
'TABLE'
>>>
```

Nếu ký tự ch là chữ thường thì chuyển sang chữ hoa.

Chú ý: Tính chất trên không còn đúng với các bảng chữ cái tiếng Việt nữa. Vì vậy khi xử lý tiếng Việt cần rất chú ý điểm này.

2. Tập hợp các ký tự và xâu ký tự sẽ có một quan hệ thứ tự mặc định theo mã hóa tương ứng của chúng, quan hệ thứ tự này được gọi là “thứ tự từ điển”. Quan hệ thứ tự từ điển được xác định như sau:

- Ký tự $ch1 < ch2$ khi và chỉ khi mã hóa của chúng có quan hệ như vậy: $ord(ch1) < ord(ch2)$.

- Xâu ký tự $S1 < S2$ nếu ký tự đầu tiên khác nhau của 2 xâu này tính từ trái thỏa mãn quan hệ đó.

Ví dụ:

- Cần so sánh ký tự "m" và "p". Tìm trên bảng mã ASCII hay hàm `ord()` ta có $m \rightarrow 109$, còn $p \rightarrow 112$, do đó $m < p$.

- Cần so sánh xâu ký tự "hoa" và "hanh": ký tự đầu tiên giống nhau, ký tự thứ 2, "o" > "a", do vậy "hoa" > "hanh".

Chương trình sau sẽ sắp xếp theo thứ tự từ điển tăng dần từ một danh sách ban đầu các tên người.

Sap_xep_DS.py

```
1. A = ["Oanh", "Thanh", "Ba", "Rinh", "Quang", "Hanh", "Vinh",  
   "Hoa", "Loan", "An"]  
2. def Sapxep(A):  
3.     DS = A.copy()  
4.     for i in range(len(DS)-1):  
5.         for j in range(i+1, len(DS)):  
6.             if DS[i] > DS[j]:  
7.                 DS[i], DS[j] = DS[j], DS[i]  
8.     return DS  
9.  
10. print("Danh sách gốc:")  
11. print(A)  
12. print("Danh sách sau khi đã sắp xếp:")  
13. print(Sapxep(A))
```

Hàm `Sapxep()` sẽ trả về danh sách đã sắp xếp của danh sách gốc.

Kết quả chạy của chương trình trên như sau:

```
Danh sách gốc:  
['Oanh', 'Thanh', 'Ba', 'Rinh', 'Quang', 'Hanh', 'Vinh', 'Hoa',  
'Loan', 'An']  
Danh sách sau khi đã sắp xếp:  
['An', 'Ba', 'Hanh', 'Hoa', 'Loan', 'Oanh', 'Quang', 'Rinh',  
'Thanh', 'Vinh']
```

3. Dãy thoát Escape

Dãy Escape hay dãy thoát dùng để chỉ các nhóm ký tự có ý nghĩa đặc biệt bên trong xâu ký tự. Các dãy này không thể hiện như các ký tự khác là “viết gì in nấy” mà chúng có ý nghĩa điều khiển việc thể hiện các xâu ký tự này.

Tất cả các dãy điều khiển này, hay còn gọi là dãy Escape đều bắt đầu từ ký tự “\”. Quan sát ví dụ sau để tìm hiểu về các dãy điều khiển này.


```
>>> s = ""Hà Nội
Việt Nam
Thủ đô ""
>>> print(s)
Hà Nội
Việt Nam
Thủ đô
>>> s
'Hà Nội\nViệt Nam\nThủ đô '
>>>
```

Trong Python Shell hay dòng lệnh, có thể dùng 3 dấu ngoặc (đơn hoặc kép để có thể nhập xâu ký tự trên nhiều dòng.

Xâu ký tự sẽ tạo ra dãy ký tự đặc biệt "\n" để điều khiển thể hiện xuống dòng.

Dãy ký tự "\n" được gọi là dãy thoát, hay dãy Escape dùng để điều khiển thể hiện xuống dòng của xâu ký tự trên màn hình. Các dãy này nằm bên trong xâu ký tự. Chúng ta có thể viết chèn trực tiếp các dãy Escape ngay bên trong xâu ký tự.

Quan sát ví dụ sau:

```
>>> s = " Sáng nay \n Trời mưa \n Mây bay"
>>> print(s)
Sáng nay
Trời mưa
Mây bay
>>>
```

Các dãy Escape

Trong bảng mã ASCII, các ký tự đánh số từ 0 đến 31 và ký tự 127 là các ký tự đặc biệt. Các ký tự này là ký tự điều khiển và không thể hiện trên màn hình như các ký tự bình thường khác. Mỗi ký tự điều khiển này đều tương ứng với một dãy Escape và có ý nghĩa riêng biệt đối với từng loại máy tính và thiết bị khác nhau. Bảng sau mô tả một số dãy Escape tương ứng với các ký tự điều khiển của bảng mã ASCII. Chú ý trong các ký tự trên chỉ có ký tự HT (\t) và LF (\n) là có ý nghĩa trực tiếp trong việc điều khiển thể hiện xâu ký tự của Python. Các ký tự điều khiển khác của bảng này chỉ tạo ra khoảng trống trong xâu ký tự, tương tự ký tự SPACE.

Mã ASCII	Ký hiệu	Tên ký tự	Dãy thoát Escape	Mô tả ý nghĩa
0	NULL	Null character	\0	
7	BEL	Bell (alert)	\a	
8	BS	Backspace	\b	
9	HT	Horizontal Tab	\t	Tạo tab sau ký tự này.
10	LF	Line Feed (new line)	\n	Tạo dòng mới (xuống dòng) sau ký tự này.
11	VT	Vertical Feed	\v	
12	FF	Form Feed	\f	
13	CR	Carriage Return	\c	

Ngoài các dãy Escape tương ứng với các ký tự điều khiển của bảng ASCII, trong Python còn có nhiều dãy Escape khác mà không liên quan đến các ký tự của bảng mã ASCII. Bảng sau liệt kê một số dãy thoát như vậy.

Bảng một số dãy Escape khác.

Dãy Escape	Ý nghĩa
\<newline>	Sau \ có thể nhấn Enter để bắt đầu <newline> từ dòng mới.
\\	Ký tự backslash \
\'	Ký tự dấu nháy đơn '
\"	Ký tự dấu nháy kép "
\N(<tên ký tự Unicode>)	Ký tự bảng mã Unicode với tên tương ứng.
\u<hhhh>	Ký tự Unicode với mã 16-bit hex <hhhh>
\x<hh>	Ký tự 8-bit (ASCII) với mã hex <hh>

Chúng ta cùng xem minh họa trong các ví dụ sau.

```
>>> s = "A! Tiếng hát\  
Ngọt như tiếng hát\  
Của các em."  
>>> s  
'A! Tiếng hátNgọt như tiếng hátCủa các em.'  
>>> s = "Đây là ký tự backslash \\  
>>> print(s)  
Đây là ký tự backslash \.  
>>> s = "I don't think so."  
>>> print(s)  
I don't think so.  
>>>
```

4. Các phương thức tự động điều chỉnh và căn chỉnh xâu ký tự

Mặc dù xâu ký tự không cho phép thay đổi trực tiếp giá trị trên từng ký tự theo truy cập chỉ số, nhưng Python cho chúng ta rất nhiều phương thức hữu ích để thay đổi, điều chỉnh nội dung của xâu. Chúng ta sẽ xem xét các phương thức đó trong phần này.

4.1. Nhóm lệnh điều chỉnh liên quan đến chữ hoa, chữ thường

Nhóm các lệnh này được xây dựng dựa trên cấu trúc mã hóa bộ chữ cái tiếng Anh trong bảng mã chuẩn ASCII. Chúng ta đã biết quy luật “cách nhau 32” giữa các chữ cái hoa và chữ thường trong bảng mã này.

Sau đây là danh sách các hàm thuộc nhóm này.

Hàm số	Mô tả ý nghĩa
upper()	Hàm này trả lại xâu sau khi đã chuyển tất cả các ký tự thành chữ hoa.
lower()	Hàm này trả lại xâu sau khi đã chuyển tất cả các ký tự thành chữ thường.

Hàm số	Mô tả ý nghĩa
<code>capitalize()</code>	Hàm này sẽ chuyển ký tự đầu tiên của xâu thành chữ hoa, còn tất cả các chữ còn lại chuyển về chữ thường.
<code>title()</code>	Hàm này sẽ biến đổi tất cả các chữ cái ở đầu các từ thành chữ hoa, còn các chữ còn lại chuyển về chữ thường.
<code>swapcase()</code>	Hàm sẽ chuyển các chữ thường thành chữ hoa và ngược lại, biến chữ hoa thành chữ thường.

Chú ý: các hàm trên đây chỉ đúng với các xâu ký tiếng Anh. Các hàm này không áp dụng cho tiếng Việt.

Sau đây là các ví dụ.

```
>>> s = "Last Friday over 4 million people striked for the CLIMATE"
>>> s.lower()
'last friday over 4 million people striked for the climate'
>>> s.upper()
'LAST FRIDAY OVER 4 MILLION PEOPLE STRIKED FOR THE CLIMATE'
>>> s.capitalize()
'Last friday over 4 million people striked for the climate'
>>> s.title()
'Last Friday Over 4 Million People Striked For The Climate'
>>> s.swapcase()
'LAST FRIDAY OVER 4 MILLION PEOPLE STRIKED FOR THE climate'
>>>
```

4.2. Nhóm lệnh tự điều chỉnh khoảng trống thừa

Các hàm số này sẽ điều chỉnh xóa các khoảng trống thừa phía trước và phía sau của một xâu ký tự. Các lệnh đơn giản này như sau.

Hàm số	Mô tả ý nghĩa
<code>strip()</code>	Hàm này sẽ trả lại xâu gốc sau khi đã xóa các ký tự trống trước và sau xâu ký tự hiện thời.
<code>lstrip()</code>	Hàm này sẽ trả lại xâu gốc sau khi đã xóa các ký tự trống phía trước xâu ký tự hiện thời.
<code>rstrip()</code>	Hàm này sẽ trả lại xâu gốc sau khi đã xóa các ký tự trống phía sau xâu ký tự hiện thời.

Sau đây là các ví dụ.

```
>>> s = "   Hà Nội   "
>>> s.lstrip()
'Hà Nội'
>>> s.rstrip()
'   Hà Nội'
>>> s.strip()
'Hà Nội'
>>>
```

Chú ý: Các lệnh trên cũng xóa các dãy Escape nằm ở phía trước và sau xâu ký tự.

```
>>> s = "quê hương \n"
>>> s
'quê hương \n'
>>> s.strip()
'quê hương'
>>> s = "\tquê hương \n"
>>> s.strip()
'quê hương'
>>>
```

4.3. Lệnh count()

Lệnh count() có chức năng đếm số lần xuất hiện không chồng lấn của một ký tự hay xâu con bên trong một xâu lớn.

Trong ví dụ sau, chữ "a" xuất hiện trong xâu "ababababab" 5 lần, còn xâu "aba" xuất hiện trong xâu trên 4 lần, nhưng nếu tính không chồng lấn thì chỉ 2 lần.

```
>>> s = "ababababab"
>>> s.count("a")
5
>>> s.count("aba")
2
```

Phương thức count() còn có thêm 2 tham số lựa chọn nữa.

Cú pháp tổng quát của hàm count() như sau:

```
count(<xâu cần tìm>, [chỉ số start, chỉ số end])
```

Lệnh trên sẽ tìm số lần lặp không chồng lên nhau của xâu con trong xâu mẹ nhưng chỉ tính trong vùng chỉ số [start : end].

Ví dụ:

```
>>> s = "00001111"
>>> s.count("0")
4
>>> s.count("0", 2)           # Tìm số lần số 0 nằm trong s[2 : ]
2
>>> s.count("1", 0, 5)        # Tìm số lần số 1 nằm trong s[0 : 5]
1
>>> s[0:5].count("1")         # lệnh này tương đương lệnh trên
1
>>>
```

Chú ý: Việc tìm kiếm này là phân biệt chữ thường chữ hoa.

4.4. Nhóm lệnh find() và index()

Nhóm lệnh tiếp theo find() và index() đều là các lệnh tìm kiếm, tương tự như count(), nhưng kết quả trả lại là chỉ số đầu tiên tìm thấy của xâu con.

Như vậy nếu tìm thấy xâu con trong xâu gốc thì cả 2 phương thức trên đều trả lại giá trị chỉ số tìm thấy, do vậy kết quả của 2 hàm sẽ như nhau. Nhưng nếu không tìm thấy xâu con thì kết quả trả lại của 2 hàm sẽ khác nhau: hàm find() sẽ trả lại giá trị -1, còn hàm

index() sẽ thông báo lỗi và biến nhớ hệ thống ValueError sẽ nhận giá trị "substring not found".

Cùng xem các ví dụ sau.

```
>>> s = "001122334455"
>>> s.find("12")
3
>>> s.index("12")
3
>>> s.find("123")
-1
>>> s.index("123")
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    s.index("123")
ValueError: substring not found
>>>
```

Không tìm thấy xâu con, hàm index() báo lỗi trong biến ValueError.

Cú pháp tổng quát của các phương thức trên như sau, tương tự như với hàm count().

```
find(<xâu cần tìm>, [chỉ số start, chỉ số end])
index(<xâu cần tìm>, [chỉ số start, chỉ số end])
```

Ý nghĩa các tham số tự chọn hoàn toàn tương tự như với hàm count().

Chú ý: Việc tìm kiếm này là phân biệt chữ thường chữ hoa.

4.5. Lệnh replace()

Lệnh replace() cho phép thay thế một cụm từ (xâu con) bằng một cụm từ khác trong xâu gốc. Việc tìm kiếm thay thế sẽ tính từ trái sang. Ví dụ:

```
>>> s = "ababa"
>>> s.replace("aba", "000")
'000ba'
>>> s.replace("a", "A")
'AbAbA'
>>>
```

Cú pháp tổng quát của lệnh này là:

```
replace(<old>, <new>, [count = -1])
```

Tìm và thay thế các xâu <old> thành <new>. Tham số lựa chọn count chỉ ra số lần thay thế lớn nhất có thể thực hiện. Nếu count = -1 (mặc định) thì không hạn chế số lần thực hiện thay thế.

Ví dụ:

```
>>> s = "ababa"
>>> s.replace("a", "A", 2)
'AbAbA'
>>> s.replace("a", "A", 1)
'Ababa'
>>>
```


Chú ý: Các lệnh trên cũng xóa các dãy Escape nằm ở phía trước và sau dấu ký tự.

```
>>> s = "quê hương \n"
>>> s
'quê hương \n'
>>> s.strip()
'quê hương'
>>> s = "\tquê hương \n"
>>> s.strip()
'quê hương'
>>>
```

4.3. Lệnh count()

Lệnh count() có chức năng đếm số lần xuất hiện không chồng lấn của một ký tự hay chuỗi con bên trong một chuỗi lớn.

Trong ví dụ sau, chữ "a" xuất hiện trong chuỗi "ababababab" 5 lần, còn chuỗi "aba" xuất hiện trong chuỗi trên 4 lần, nhưng nếu tính không chồng lấn thì chỉ 2 lần.

```
>>> s = "ababababab"
>>> s.count("a")
5
>>> s.count("aba")
2
```

Phương thức count() còn có thêm 2 tham số lựa chọn nữa.

Cú pháp tổng quát của hàm count() như sau:

```
count(<chuỗi cần tìm>, [chỉ số start, chỉ số end])
```

Lệnh trên sẽ tìm số lần lặp không chồng lấn nhau của chuỗi con trong chuỗi mẹ nhưng chỉ tính trong vùng chỉ số [start : end].

Ví dụ:

```
>>> s = "00001111"
>>> s.count("0")
4
>>> s.count("0", 2)           # Tìm số lần số 0 nằm trong s[2 : ]
2
>>> s.count("1", 0, 5)        # Tìm số lần số 1 nằm trong s[0 : 5]
1
>>> s[0:5].count("1")         # lệnh này tương đương lệnh trên
1
>>>
```

Chú ý: Việc tìm kiếm này là phân biệt chữ thường chữ hoa.

4.4. Nhóm lệnh find() và index()

Nhóm lệnh tiếp theo find() và index() đều là các lệnh tìm kiếm, tương tự như count(), nhưng kết quả trả lại là chỉ số đầu tiên tìm thấy của chuỗi con.

Như vậy nếu tìm thấy chuỗi con trong chuỗi gốc thì cả 2 phương thức trên đều trả lại giá trị chỉ số tìm thấy, do vậy kết quả của 2 hàm sẽ như nhau. Nhưng nếu không tìm thấy chuỗi con thì kết quả trả lại của 2 hàm sẽ khác nhau: hàm find() sẽ trả lại giá trị -1, còn hàm

index() sẽ thông báo lỗi và biến nhớ hệ thống ValueError sẽ nhận giá trị "substring not found".

Cùng xem các ví dụ sau.

```
>>> s = "001122334455"
>>> s.find("12")
3
>>> s.index("12")
3
>>> s.find("123")
-1
>>> s.index("123")
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    s.index("123")
ValueError: substring not found
>>>
```

Không tìm thấy xâu con, hàm index() báo lỗi trong biến ValueError.

Cú pháp tổng quát của các phương thức trên như sau, tương tự như với hàm count().

```
find(<xâu cần tìm>, [chỉ số start, chỉ số end])
index(<xâu cần tìm>, [chỉ số start, chỉ số end])
```

Ý nghĩa các tham số tự chọn hoàn toàn tương tự như với hàm count().

Chú ý: Việc tìm kiếm này là phân biệt chữ thường chữ hoa.

4.5. Lệnh replace()

Lệnh replace() cho phép thay thế một cụm từ (xâu con) bằng một cụm từ khác trong xâu gốc. Việc tìm kiếm thay thế sẽ tính từ trái sang. Ví dụ:

```
>>> s = "ababa"
>>> s.replace("aba", "000")
'000ba'
>>> s.replace("a", "A")
'AbAbA'
>>>
```

Cú pháp tổng quát của lệnh này là:

```
replace(<old>, <new>, [count = -1])
```

Tìm và thay thế các xâu <old> thành <new>. Tham số lựa chọn count chỉ ra số lần thay thế lớn nhất có thể thực hiện. Nếu count = -1 (mặc định) thì không hạn chế số lần thực hiện thay thế.

Ví dụ:

```
>>> s = "ababa"
>>> s.replace("a", "A", 2)
'AbAbA'
>>> s.replace("a", "A", 1)
'AbabA'
>>>
```

4.6. Lệnh split() và join()

Chúng ta sẽ xem xét 2 phương thức cuối cùng trong phần này, đó là các hàm split() và join(). Các hàm này có ý nghĩa ngược nhau.

Hàm **split()** sẽ tách một chuỗi ký tự thành một danh sách các chuỗi con. Mặc định danh sách này là các từ đơn của chuỗi gốc sau khi loại bỏ các dấu trống và dãy Escape thừa bên trong chuỗi này.

Hàm **join()** có ý nghĩa ngược lại. Đầu vào của hàm là một list các dữ liệu kiểu string. Hàm này sẽ nối các phần tử này lại để tạo thành một chuỗi ghép từ các thành phần trên. Chuỗi ký tự gọi hàm có ý nghĩa làm ngăn cách giữa các phần tử khi ghép nối.

Ví dụ với hàm split().

```
>>> s = "Hà Nội là thủ đô của nước Việt Nam"
>>> s
'Hà Nội là thủ đô của nước Việt Nam'
>>> s.split()
['Hà', 'Nội', 'là', 'thủ', 'đô', 'của', 'nước', 'Việt', 'Nam']
>>>
```

Trong ví dụ trên, chuỗi gốc s được nhập có thừa nhiều dấu cách. Kết quả của hàm split() là dãy (list) các thành phần, từ đơn của chuỗi này.

Ví dụ sau thực hiện quá trình ngược lại, dùng hàm join().

```
>>> a = ['Hà', 'Nội', 'là', 'thủ', 'đô', 'của', 'nước', 'Việt', 'Nam']
>>> " ".join(a)
'Hà Nội là thủ đô của nước Việt Nam'
```

Xâu gốc gọi hàm join() là dấu cách " ".

Chú ý chuỗi gốc, nơi gọi hàm join() chính là chuỗi dùng làm ngăn cách khi ghép nối.

Cú pháp tổng quát của lệnh split() là:

```
split([sep = none, maxsplit = -1])
```

Trong đó 2 tham số lựa chọn có ý nghĩa như sau:

sep = chính là đối tượng dùng làm tiêu chuẩn tách. Mặc định sep = none, tức là không có mặt tham số này trong lệnh thì hàm sẽ lấy ký tự ngăn cách là các dấu cách và dãy Escape.

maxsplit = số lượng thành phần sẽ được tách bởi lệnh này. Nếu giá trị này nhỏ hơn số lượng từ có thể tách thì hàm sẽ trả lại phần tử cuối cùng của list là phần còn lại chưa tách của chuỗi gốc.

Chúng ta cùng quan sát các ví dụ sau.

```
1. >>> s.split(" ")
2. ['Hà', '', '', 'Nội', '', '', 'là', '', '', 'thủ', 'đô', 'của', 'nước', 'Việt', '', '', 'Nam']
3. >>> s.split(4)
4. Traceback (most recent call last):
5.   File "<pyshell#45>", line 1, in <module>
6.     s.split(4)
7. TypeError: must be str or None, not int
8. >>> s.split(" ",4)
9. ['Hà', '', '', 'Nội', ' là thủ đô của nước Việt Nam']
10. >>> s.split(maxsplit = 4)
```

```
11. ['Hà', 'Nội', 'là', 'thủ', 'đô của nước Việt Nam']
12. >>> s.split(maxsplit = 2)
13. ['Hà', 'Nội', 'là thủ đô của nước Việt Nam']
```

Nhận xét:

- Dòng lệnh 1, 2 mô tả kết quả của lệnh tách nếu dùng SPACE làm ký tự ngăn cách. Kết quả sẽ không được như mong đợi.
- Dòng lệnh 3 lỗi vì tham số đầu tiên bắt buộc phải là kiểu xâu ký tự.
- Dòng lệnh 8 dùng SPACE làm ký tự ngăn cách và chỉ lấy 4 từ tách, kết quả thu được 4 từ tách đó là "Hà", " ", " ", "Nội".
- Các dòng lệnh 10, 12 cho ta kết quả chính xác hơn khi dùng mặc định cho xâu ngăn cách.

Cú pháp tổng quát của lệnh join() là:

```
<string kết nối>.join(<dữ liệu tuần tự>)
```

Ví dụ:

```
>>> ".".join("abcde")
'a.b.c.d.e'
>>> "-".join(("1", "2", "3", "4"))
'1-2-3-4'
>>>
```

Chúng ta cùng xây dựng một ứng dụng hữu ích đơn giản.

Hàm số sau có chức năng tách một xâu ký tự bất kỳ thành các từ đơn sau khi xóa tất cả các dấu cách thừa trước, trong và sau xâu ký tự.

```
def fixTyping(s):
    a = s.split()
    return " ".join(a)
```

Cách thực hiện được chia làm 2 bước:

- Bước 1, tách xâu ký tự s thành dãy list các từ đơn bằng phương thức split().
- Bước 2, ghép các từ đơn lại bằng hàm join().

Chương trình hoàn chỉnh có thể như sau:

fixTyping.py

```
1. def fixTyping(s):
2.     a = s.split()
3.     return " ".join(a)
4.
5. s = input("Nhập một xâu bất kỳ: ")
6. print("Xâu đã sửa lỗi là:")
7. print(fixTyping(s))
```

Kết quả chạy chương trình trên có thể như sau.

```
Nhập một xâu bất kỳ: Hà Nội là thủ đô của Việt Nam
Xâu đã sửa lỗi là:
Hà Nội là thủ đô của Việt Nam
```

Phương thức `split()` còn có một ứng dụng nữa: cho phép nhập nhiều giá trị số trên một dòng nhập liệu duy nhất. Sau đây là chương trình đơn giản minh họa.

Nhap_2_so.py

```
1. s = input("Nhập 2 số m, n cách nhau bởi dấu cách: ")
2. x, y = s.split()
3. m,n = eval(x), eval(y)
4. print("Các số đã nhập là:",m,n)
```

4.7. Hàm `__repr__()`

Hàm `__repr__()` có ở mọi đối tượng trong Python. Hàm này luôn trả lại dưới dạng xâu ký tự hình ảnh gốc của đối tượng. Hàm này có ý nghĩa tham khảo rất tốt trong nhiều trường hợp. Sau đây là các ví dụ.

```
>>> a = [1,2,3,4]
>>> a.__repr__()
'[1, 2, 3, 4]'
>>> s = "Hà Nội \
là \
thủ đô"
>>> s.__repr__()
'"Hà Nội là thủ đô"'
>>> b = 12 > 6
>>> b.__repr__()
'True'
>>> x = 12.5*34
>>> x.__repr__()
'1.9721522630525294e+37'
>>>
```

5. Lệnh tạo khuôn `format()`

Lệnh tạo khuôn xâu ký tự `format()` là một trong các lệnh hay dùng nhất và phức tạp nhất của Python. Lệnh này có chức năng tạo khuôn cho việc thể hiện các xâu ký tự trên màn hình hoặc ghi ra tệp, vì vậy nó được áp dụng rất nhiều.

Việc tạo khuôn xâu ký tự bằng lệnh `format()` trong Python liên quan mật thiết đến các ký hiệu ngoặc nhọn {}, các vị trí này trong xâu được gọi là các trường (field).

Ví dụ với xâu ký tự sau "Xin chào {} và {}". Đây là xâu ký tự hoàn toàn bình thường.

```
>>> s = "Xin chào {} và {}"
>>> s.__repr__()
'"Xin chào {} và {}"'
>>> print(s)
Xin chào {} và {}
```

Trong ví dụ trên các ký tự {, } và các vị trí {} không có ý nghĩa gì đặc biệt trong xâu, việc thể hiện chúng vẫn như bình thường.

Tuy nhiên ý nghĩa của các trường này sẽ trở nên đặc biệt trong ví dụ sau khi đưa thêm hàm format() vào.

```
>>> s = "Xin chào {} và {}"  
>>> s.format("Hòa", "Bình")  
'Xin chào Hòa và Bình'  
>>>
```

Chúng ta thấy gì?

- Các vị trí của xâu có chứa {} nay đã thay thế bằng dữ liệu cụ thể, các dữ liệu này có trong tham số của hàm format().

Như vậy các vị trí {} đóng vai trò rất quan trọng trong xâu ký tự, nó làm thay đổi nội dung thể hiện của xâu trong lệnh tạo khuôn format(). Các vị trí này {} được gọi là các **trường**, hay **field**.



```
s = "Xin chào {} và {}"
```

Các vị trí **field** này sẽ đóng vai trò như các “hộp giữ chỗ” để có thể thay thế dữ liệu thông qua hàm format(). Chúng ta sẽ cùng tìm hiểu các “hộp giữ chỗ” này thông qua lệnh tạo khuôn format() của Python.

Chúng ta sẽ thống nhất sử dụng các từ như **field**, **hộp**, **hộp giữ chỗ**, **placeholder** để chỉ các vị trí cần tạo khuôn {}.

5.1. Thay thế đơn giản không chỉ số

Trường hợp đơn giản nhất khi các hộp giữ chỗ đơn giản chỉ có dạng {}.

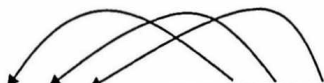
Cú pháp đơn giản là:

```
<Field> = {}
```

Ví dụ:

```
s = "Đây là các số {} {} {}"
```

Khi đó lệnh format() sẽ phải có đủ các tham số để thay thế các giá trị vào các vị trí thay thế trên, tính lần lượt từ trái sang phải.



```
>>> "Đây là các số {} {} {}".format(1, 2, 3)  
'Đây là các số 1 2 3'
```

Chú ý là các tham số của format() phải có số lượng giá trị lớn hơn hoặc bằng số lượng các vị trí thay thế. Có thể nhiều hơn chứ không thể ít hơn.

```
>>> "Đây là các số {} {}".format(1, 2, 3)
'Đây là các số 1 2 '
>>> "Đây là các số {} {}".format(1, 2)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    "Đây là các số {} {}".format(1, 2)
IndexError: tuple index out of range
>>>
```

Lệnh `format()` có số tham số nhiều hơn vị trí thay thế `{}` → OK.

Lệnh `format()` có số tham số ít hơn số vị trí thay thế `{}` → phát sinh lỗi.

Dữ liệu được thay thế trong các hộp giữ chỗ có thể là dữ liệu bất kỳ, có các kiểu dữ liệu khác nhau.

```
>>> s = "Đây là số nguyên {}, số thực {}, list {}"
>>> s.format(100, 3.1416, [1, 2, 3, "One"])
'Đây là số nguyên 100, số thực 3.1416, list [1, 2, 3, 'One']'
>>>
```

Chú ý trong hộp giữ chỗ không được phép có dấu cách thừa. Các thông tin ghi bên trong trường này phải được mô tả tuyệt đối chính xác.

Ví dụ sau cho ta thấy chương trình sẽ báo lỗi nếu để thừa dấu cách bên trong dấu `{}`.

```
>>> s = " Số { } "
>>> s.format(3)
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    s.format(3)
KeyError: ' '
>>>
```

Sau đây là một ứng dụng nhỏ của phương thức `format()` với xâu ký tự cùng các hộp giữ chỗ đơn giản (không chỉ số).

Giả sử chúng ta có một danh sách học sinh, khi đó thể in danh sách lớp có thể thực hiện bằng cách sau.

```
1. s = "Học sinh thứ {} tên là {}"
2. HS = ["Hà", "Bình", "Quang", "Vinh", "Việt", "Cường"]
3. for i in range(len(HS)):
4.     print(s.format(i+1, HS[i]))
```

Kết quả chương trình trên như sau.

```
Học sinh thứ 1 tên là Hà
Học sinh thứ 2 tên là Bình
Học sinh thứ 3 tên là Quang
Học sinh thứ 4 tên là Vinh
Học sinh thứ 5 tên là Việt
Học sinh thứ 6 tên là Cường
```

5.2. Thay thế tự động với chỉ số

Trong phần trên chúng ta đã biết cách tạo khuôn mẫu ký tự với các hộp {} không chỉ số. Khi đó việc áp dụng thay thế sẽ tính từ trái sang phải.

Các đối tượng dữ liệu là tham số của lệnh format() sẽ được đánh chỉ số từ 0, tính từ trái sang phải như sau.

```
format(<object-0>, <object-1>, ..., <object-N>)
```

Cách sử dụng tiếp theo của các hộp giữ chỗ là dùng chỉ số. Chỉ số sẽ được ghi rõ bên trong dấu {} để cho Python hiểu và thay thế chính xác các đối tượng dữ liệu từ danh sách tham số của mình.

Cú pháp của trường ghi chỉ số, các chỉ số đánh số từ 0.

```
<Field> = {<Chỉ số>}
```

Với cách dùng chỉ số này, việc dùng hộp giữ chỗ sẽ thuận tiện, hợp lý và rõ ràng hơn rất nhiều.

Ví dụ sau cho chúng ta thấy cách dùng các trường thay thế với chỉ số.

```
>>> m = 15
>>> n = 66
>>> "Cho trước các số {0}, {1}. Cần tìm các ước số chung của {0}
và {1}".format(m,n)
'Cho trước các số 15, 66. Cần tìm các ước số chung của 15 và 66'
>>>
```

Chúng ta thấy trong mẫu ký tự gốc, các trường {<chỉ số>} có thể lặp lại nhiều lần mà lệnh format() vẫn thay thế chính xác, điều này có được nhờ chính các chỉ số này.

Chú ý nếu đã dùng chỉ số thì phải dùng toàn bộ với các hộp giữ chỗ, không được phép dùng xem kẽ không và có chỉ số.

```
>>> "{0} {1} {}".format(0, 1, 2)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    "{0} {1} {}".format(0, 1, 2)
ValueError: cannot switch from manual field specification to
automatic field numbering
```

5.3. Chỉ lệnh trong field

Bên trong mỗi hộp giữ chỗ {}, ngoài chỉ số mà ta đã biết, Python còn cho phép đưa vào thêm các ký tự điều khiển khác nữa. Các ký tự điều khiển này được gọi là các chỉ lệnh. Các chỉ lệnh sẽ được viết bên phải chỉ số, sau dấu ":". Cú pháp của chỉ lệnh như hình sau.

Ký tự : ngăn cách giữa chỉ số
và chỉ lệnh.

```
<Field> = {<Chỉ số>:<chỉ lệnh>}
```

Vì chỉ số và chỉ lệnh trong hộp giữ chỗ đều là lựa chọn, có thể có hoặc không, do đó tổng quát có thể có các cách viết sau.

Lệnh format() có số tham số nhiều hơn vị trí thay thế {} → OK.

```
>>> "Đây là các số {} {}".format(1, 2, 3)
'Đây là các số 1 2 '
```

Lệnh format() có số tham số ít hơn số vị trí thay thế {} → phát sinh lỗi.

```
>>> "Đây là các số {} {} {}".format(1, 2)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    "Đây là các số {} {} {}".format(1, 2)
IndexError: tuple index out of range
>>>
```

Dữ liệu được thay thế trong các hộp giữ chỗ có thể là dữ liệu bất kỳ, có các kiểu dữ liệu khác nhau.

```
>>> s = "Đây là số nguyên {}, số thực {}, list {}"
>>> s.format(100, 3.1416, [1, 2, 3, "One"])
'Đây là số nguyên 100, số thực 3.1416, list [1, 2, 3, 'One']'
>>>
```

Chú ý trong hộp giữ chỗ không được phép có dấu cách thừa. Các thông tin ghi bên trong trường này phải được mô tả tuyệt đối chính xác.

Ví dụ sau cho ta thấy chương trình sẽ báo lỗi nếu để thừa dấu cách bên trong dấu {}.

```
>>> s = " Số { }"
>>> s.format(3)
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    s.format(3)
KeyError: ' '
```

Sau đây là một ứng dụng nhỏ của phương thức format() với dấu ký tự cùng các hộp giữ chỗ đơn giản (không chỉ số).

Giả sử chúng ta có một danh sách học sinh, khi đó thể in danh sách lớp có thể thực hiện bằng cách sau.

```
1. s = "Học sinh thứ {} tên là {}"
2. HS = ["Hà", "Bình", "Quang", "Vinh", "Việt", "Cường"]
3. for i in range(len(HS)):
4.     print(s.format(i+1, HS[i]))
```

Kết quả chương trình trên như sau.

```
Học sinh thứ 1 tên là Hà
Học sinh thứ 2 tên là Bình
Học sinh thứ 3 tên là Quang
Học sinh thứ 4 tên là Vinh
Học sinh thứ 5 tên là Việt
Học sinh thứ 6 tên là Cường
```


5.2. Thay thế tự động với chỉ số

Trong phần trên chúng ta đã biết cách tạo khuôn mẫu ký tự với các hộp {} không chỉ số. Khi đó việc áp dụng thay thế sẽ tính từ trái sang phải.

Các đối tượng dữ liệu là tham số của lệnh format() sẽ được đánh chỉ số từ 0, tính từ trái sang phải như sau.

```
format(<object-0>, <object-1>, ..., <object-N>)
```

Cách sử dụng tiếp theo của các hộp giữ chỗ là dùng chỉ số. Chỉ số sẽ được ghi rõ bên trong dấu {} để cho Python hiểu và thay thế chính xác các đối tượng dữ liệu từ danh sách tham số của mình.

Cú pháp của trường ghi chỉ số, các chỉ số đánh số từ 0.

```
<Field> = {<Chỉ số>}
```

Với cách dùng chỉ số này, việc dùng hộp giữ chỗ sẽ thuận tiện, hợp lý và rõ ràng hơn rất nhiều.

Ví dụ sau cho chúng ta thấy cách dùng các trường thay thế với chỉ số.

```
>>> m = 15
>>> n = 66
>>> "Cho trước các số {0}, {1}. Cần tìm các ước số chung của {0}
và {1}".format(m,n)
'Cho trước các số 15, 66. Cần tìm các ước số chung của 15 và 66'
>>>
```

Chúng ta thấy trong mẫu ký tự gốc, các trường {<chỉ số>} có thể lặp lại nhiều lần mà lệnh format() vẫn thay thế chính xác, điều này có được nhờ chính các chỉ số này.

Chú ý nếu đã dùng chỉ số thì phải dùng toàn bộ với các hộp giữ chỗ, không được phép dùng xem kẽ không và có chỉ số.

```
>>> "{0} {1} {}".format(0, 1, 2)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    "{0} {1} {}".format(0, 1, 2)
ValueError: cannot switch from manual field specification to
automatic field numbering
```

5.3. Chỉ lệnh trong field

Bên trong mỗi hộp giữ chỗ {}, ngoài chỉ số mà ta đã biết, Python còn cho phép đưa vào thêm các ký tự điều khiển khác nữa. Các ký tự điều khiển này được gọi là các chỉ lệnh. Các chỉ lệnh sẽ được viết bên phải chỉ số, sau dấu ":". Cú pháp của chỉ lệnh như hình sau.

Ký tự : ngăn cách giữa chỉ số
và chỉ lệnh.

```
<Field> = {<Chỉ số>:<chỉ lệnh>}
```

Vì chỉ số và chỉ lệnh trong hộp giữ chỗ đều là lựa chọn, có thể có hoặc không, do đó tổng quát có thể có các cách viết sau.

Dạng đầy đủ cả chỉ số và chỉ lệnh:

```
<Field> = {<Chỉ số>:<Chỉ lệnh>}
```

Dạng thiếu:

```
<Field> = {} # field rỗng không có chỉ số.  
<Field> = {<Chỉ số>} # có chỉ số, không chỉ lệnh.  
<Field> = {:<Chỉ lệnh>} # không có chỉ số, có chỉ lệnh.
```

Chỉ lệnh sẽ điều khiển sâu hơn cách thể hiện các dữ liệu thay thế, trong phần tiếp theo chúng ta sẽ tìm hiểu về các chỉ lệnh này.

5.4. Chỉ lệnh thiết lập chiều rộng Width

Nếu chỉ lệnh là một số tự nhiên lớn hơn 0 thì đó là chỉ lệnh qui định độ rộng của tham số cần hiển thị. Trên thực tế rất nhiều khi chúng ta cần qui định dữ liệu hiển thị ra thẳng hàng theo các cột chữ hoặc số, khi đó chỉ lệnh chiều rộng dữ liệu sẽ cần đến.

Cú pháp của chỉ lệnh này như sau:

Chỉ lệnh là một số tự nhiên **Width** > 0, qui định khoảng rộng thể hiện.

```
<Field> = {<Chỉ số>:<Width>}
```

Trong ví dụ sau, xâu cần hiển thị có 3 trường dữ liệu cùng có chỉ số lệnh chiều rộng = 5. Chúng ta sẽ thấy các dữ liệu số và xâu ký tự được hiển thị khác nhau.

```
>>> s = "Các giá trị a = {:5} b = {:5} c = {:5}"  
>>> s.format(1, 3, 6)  
'Các giá trị a =      1 b =      3 c =      6'  
>>> s.format(123, 1567, 11011)  
'Các giá trị a =    123 b =   1567 c = 11011'  
>>> s.format("Hà", "Bình", "Ca")  
'Các giá trị a = Hà    b = Bình  c = Ca    '  
>>>
```

Mặc định, dữ liệu số sẽ căn phải bên trong vùng thể hiện, xâu ký tự sẽ căn trái trong vùng quy định.

Chương trình sau là một ứng dụng đơn giản để hiển thị ma trận 3 x 3 trên màn hình, sử dụng các chỉ lệnh thiết lập độ rộng của hàm format().

In_ma_tran.py

```
1. A = [[10, 12, 21], [1, 17, 100], [210, 420, 99]]  
2. s = "{:5} {:5} {:5}" # thiết lập độ rộng các cột  
3. for item in A:  
4.     t0 = item[0]; t1 = item[1]; t2 = item[2]  
5.     print(s.format(t0, t1, t2))
```

Kết quả chạy chương trình trên như sau.

10	12	21
1	17	100
210	420	99

5.5. Chỉ lệnh căn chỉnh thẳng hàng

Khi thiết lập chỉ lệnh độ rộng trường, dữ liệu sẽ được thể hiện bên trong khoảng rộng đó nên sẽ thừa ra các khoảng trống. Ở phần trên đã chỉ ra, Python sẽ mặc định căn chỉnh dữ liệu giữa các khoảng trống đó như sau:

- Với dữ liệu số sẽ căn thẳng bên phải.
- Với xâu ký tự thì sẽ căn thẳng bên trái.

Chúng ta có thể điều khiển việc căn hàng giữa các độ rộng đó bằng các ký hiệu bổ sung vào chỉ lệnh.

Cú pháp của các chỉ lệnh căn hàng như sau:

Dấu = đứng sát phía trước **Width** qui định cách căn chỉnh hàng của dữ liệu.

<Field> = {<Chỉ số>:=<Width>}

Cho phép các ký hiệu căn chỉnh sau:

= căn đều 2 bên.

^ căn giữa.

> căn phải.

< căn trái.

Ví dụ:

```
>>> s = "{0:<10} {0:^10} {0:>10}"
>>> s.format(1234)
'1234          1234          1234'
>>> s.format("Hà Nội")
'Hà Nội      Hà Nội      Hà Nội'
>>>
```

5.6. Chỉ lệnh chèn ký tự vào khoảng trống

Các khoảng trống thừa khi thể hiện dữ liệu bên trong một độ rộng nào đó có thể được thay thế không phải là ký tự trống, mà có thể ký tự bất kỳ.

Với các chỉ lệnh này, ký tự sẽ được chèn vào các khoảng trống sẽ đặt phía trước của chỉ lệnh độ rộng Width, và trước cả chỉ lệnh căn hàng nếu có.

Cú pháp của chỉ lệnh này như sau.

Ký tự sẽ được chèn vào các khoảng trống.

<Field> = {<Chỉ số>:<char>=<Width>}

= căn đều 2 bên.

< căn trái.

> căn phải.

^ căn giữa

Trong ví dụ sau, các ký tự "-" sẽ được chèn vào các vị trí cần trái hoặc phải, còn ký tự "*" sẽ được chèn vào vị trí cần giữa.

```
>>> s = "{0:<10} {0:*^10} {0:>10}"
>>> s.format(123)
'123----- **123*** -----123'
>>>
```

Chú ý: Các ký tự tham gia chỉ lệnh ký tự chèn và chỉ lệnh căn hàng chỉ có ý nghĩa khi chỉ lệnh Width > 0. Nếu không có chỉ lệnh độ rộng Width thì các chỉ lệnh khác như căn hàng hay ký tự chèn sẽ trở nên vô nghĩa.

Ví dụ:

```
>>> s = "{:*>}"
>>> s.format(123)
'123'
>>> s.format("Hà Nội")
'Hà Nội'
>>>
```

5.7. Chỉ lệnh thể hiện số gần đúng (Precision, Maximum Width)

Chỉ lệnh tiếp theo hay được dùng và rất cần thiết để thể hiện các số thập phân với nhiều chữ số sau dấu phẩy.

Ví dụ số 3.141592653589793 có quá nhiều chữ số sau dấu phẩy, vì vậy khi thể hiện cần hạn chế các chữ số này, ví dụ chỉ yêu cầu thể hiện 3.14 thôi.

Khi đó chúng ta cần một chỉ lệnh mới để điều khiển số các chữ số thập phân, bao gồm các số đứng trước và sau dấu phẩy của số thập phân sẽ được thể hiện trên màn hình. Chỉ lệnh này sẽ áp dụng cho các đối tượng dữ liệu số.

Cú pháp của chỉ lệnh này như sau:

dấu chấm ngăn cách

<Field> = {<Chi số>:<Width>.<p>}

p = số các chữ số thập phân.

Width = độ rộng thể hiện của số.
Nếu giá trị này không có thì sẽ không có chỉ lệnh chiều rộng.

Ví dụ:

```
>>> import math
>>> s = "Số Pi: {:.10.3}. Số e: {:.10.3}"
>>> s.format(math.pi, math.e)
'Số Pi:      3.14. Số e:      2.72'
>>>
```

Nếu đối tượng số thập phân có dạng 0.xxxx thì chỉ lệnh <p> = số các số sau dấu phẩy, còn nếu số thập phân > 1 thì <p> = tổng số các chữ số trước và sau dấu phẩy.

```
>>> s = "{:.5}"
>>> s.format(1.344677777)
'1.3447' # chỉ hiện 4 chữ số sau dấu phẩy.
>>> s.format(0.1234567)
'0.12346' # hiện đủ 5 chữ số sau dấu phẩy
>>>
```

Chú ý: Chỉ lệnh <p> được thiết kế dành cho biểu diễn số thập phân với dấu phẩy động. Tuy nhiên chỉ lệnh này cũng có thể áp dụng cho kiểu xâu ký tự. Khi đó giá trị <p> có ý nghĩa là thiết lập độ rộng maximum thể hiện của xâu, nếu xâu dài quá sẽ bị cắt bớt.

Ví dụ:

```
>>> s = " {:<10.5} {:<10.5} {:<10.5} "
>>> s.format(0.3456789, "Hà Nội", "Việt Nam")
' 0.34568      Hà Nội      Việt      '
```

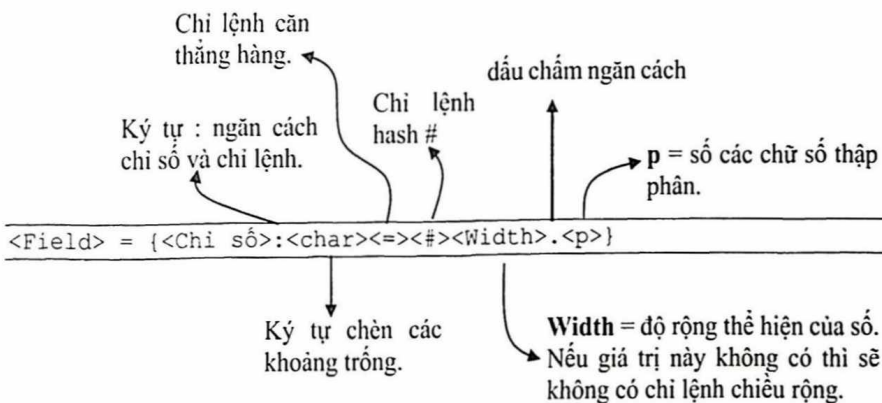
5.8. Chỉ lệnh # (hash)

Chỉ lệnh # được viết trước chỉ lệnh <Width>. <p> sẽ có tác dụng điền thêm các số 0 bên phải để số thập phân có đủ chiều dài <p>.

Ví dụ:

```
>>> s = "{:10.5} {:#10.5}"
>>> s.format(0.12, 0.4)
'      0.12      0.40000'
>>>
```

Như vậy cú pháp hoàn chỉnh của chỉ lệnh liên quan đến độ rộng thể hiện là:



Chú ý: chỉ số hash có thể đồng hành cùng các chỉ số căn thẳng hàng.

```
>>> s = "{:<10.5} {:#<10.5}"
>>> s.format(0.12, 0.4)
'0.12      0.40000      '
>>>
```

5.9. Chỉ lệnh kiểu (type)

Cuối cùng là chỉ định kiểu. Chỉ định này là một trong các ký tự **b, c, d, f, e, g** và nằm ở cuối phần chỉ lệnh dùng để quy định cách thể hiện của các kiểu dữ liệu số khác nhau. Ý nghĩa các ký hiệu chỉ lệnh kiểu này như sau:

b - biểu diễn số nhị phân.

c - biểu diễn ký tự tương ứng của bảng mã chuẩn (ASCII hoặc Unicode).

d - biểu diễn số nguyên.

f - số thập phân với số các chữ số thập phân cố định.

e - biểu diễn số thập phân dưới dạng lũy thừa của số **e**.

g - biểu diễn số dưới dạng tổng quát nhất. Python sẽ tự chọn khuôn dạng thể hiện.

Sau đây là các ví dụ minh họa.

```
>>> s = "{0:b} {0:c} {0:d}"
>>> s.format(66)
'1000010 B 66'
s1 = "{0:b} {0:d} {0:f} {0:e}"
>>> s1.format(65.2)
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    s1.format(65.2)
ValueError: Unknown format code 'b' for object of type 'float'
>>> s1.format(65)
'1000001 65 65.000000 6.500000e+01'
```

5.10. Chỉ lệnh đặt dấu phẩy (,) vào các nhóm 3 chữ số

Chỉ lệnh này sẽ quy định cách thể hiện các số nguyên và thập phân theo từng nhóm 3 chữ số ngăn cách bởi dấu phẩy.

Ví dụ cách viết số của một số nước phương Tây và Mỹ thường như sau:

1,000,000

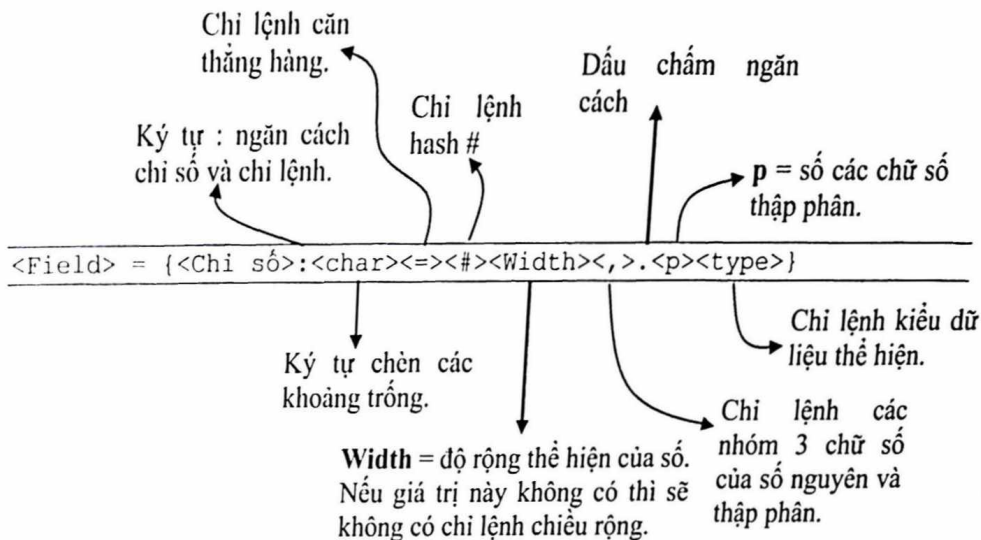
12,234,000.124,100

Chỉ lệnh này được viết trước dấu chấm (.).

```
>>> s = "{0:,.5e} {0:,.5f}"
>>> s.format(100022800.234555)
'1.00023e+08 100,022,800.23456'
>>>
```

5.11. Tóm tắt các chỉ lệnh của format()

Phương thức `format()` là lệnh tạo khuôn quan trọng nhất của xâu ký tự. Sử dụng phương thức này trên thực tế để giải các bài toán, yêu cầu khác nhau khi cần thể hiện dữ liệu số và chữ theo khuôn dạng định trước. Việc tạo khuôn dạng thông qua các hộp chỉ số và các chỉ lệnh kèm theo. Sau đây là sơ đồ tóm tắt khuôn dạng của hộp giữ số và các chỉ lệnh đã được học.



5.12. Lệnh format lỗi của Python

Trong phần trên chúng ta đã học cách tạo khuôn mẫu ra cho chuỗi ký tự bằng phương thức `format()`. Khuôn mẫu chung thực hiện các lệnh tạo khuôn như sau:

<chuỗi ký tự điều khiển>.format(<các giá trị>)

Trong đó:

<chuỗi ký tự điều khiển> là chuỗi ký tự chứa các vùng chỉ lệnh điều khiển cho lệnh `format()`.

<các giá trị> là dãy các đối tượng, giá trị cần được tạo khuôn bởi các chỉ lệnh có trong chuỗi ký tự điều khiển.

Python còn hỗ trợ cách viết khác, trực tiếp hơn bằng lệnh lỗi `format()`, chạy trực tiếp như một hàm số của Python. Lệnh `format()` sẽ trả lại chuỗi kết quả đã được tạo khuôn theo ý muốn của người sử dụng. Cú pháp của lệnh `format` như sau:

format(<giá trị>, <chuỗi chỉ lệnh điều khiển>)

Trong đó:

<giá trị> là đối tượng (số hoặc chuỗi ký tự) cần được tạo khuôn.

<chuỗi chỉ lệnh điều khiển> là chuỗi ký tự chỉ lệnh có ý nghĩa như đã mô tả trong phần trên.

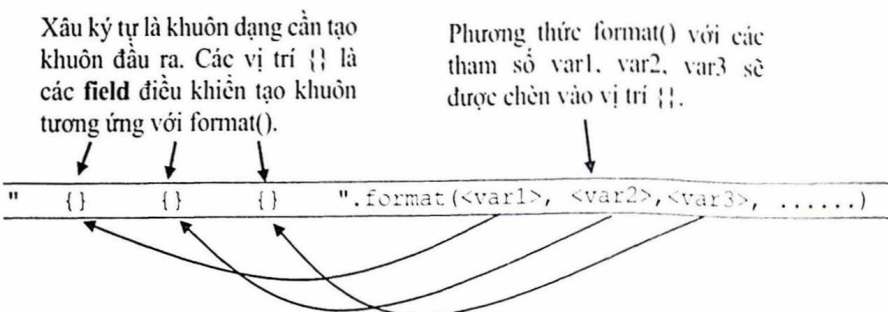
Ví dụ:

```
>>> format(123, "b")           # biểu diễn thành số nhị phân
'1111011'
>>> format("Hà Nội", ">15") # căn phải với chiều rộng 15
'          Hà Nội'
>>> format(0.123456789, "0<10.3") # căn trái với 3 chữ số thập phân
'0.12300000'
>>> format(15, "X")           # biểu diễn số dưới dạng hex
'F'
>>> format(145, "f")           # biểu diễn số thập phân
'145.000000'
>>> format(1234567890, ",,f") # số thập phân với dấu ngăn cách ",",
'1,234,567,890.000000'
```

5.13. Tạo khuôn bằng f-string

Ngoài 2 cách tạo khuôn xâu ký tự thông qua phương thức `format()` và lệnh `format()` lỗi, Python còn hỗ trợ thêm một cách làm nữa, đơn giản hơn, nhanh hơn tùy hơi “kì dị”, đó là cách tạo khuôn thông qua **f-string**.

Chúng ta nhớ lại cú pháp chính của phương thức `format()` như sau:



Trong cấu trúc trên, các biến nhớ cần tạo khuôn `<var1>`, `<var2>`, `<var3>` được đặt trong hàm `format()` và gắn với xâu điều khiển như một phương thức.

Một cách khác làm đơn giản quá trình trên là dùng **f-string**. Quan sát cấu trúc **f-string** sau sẽ thấy quan hệ giữa **f-string** và phương thức `format()` trên xâu ký tự.

```
s = f"<var1>:<chi_lệnh> {<var2>:<chi_lệnh>} {<var3>:<chi_lệnh>}"
```

Như vậy, **f-string** là một dạng xâu đặc biệt, bắt đầu bằng chữ `f`, có dạng `f<string>`, xâu sẽ chứa các trường điều khiển, mỗi trường điều khiển phải có dạng `{<var>:<chi_lệnh>}`.

Ví dụ sau minh họa cho **f-string**:

```
>>> hoten = "Bùi Việt Hà"
>>> diem = 8.9
>>> dt = "0989567767"
>>> s = f"Họ tên: {hoten:<15>} điểm: {diem:5.2f} điện thoại: {dt:>10}"
>>> s
'Họ tên: Bùi Việt Hà      điểm:  8.90 điện thoại: 0989567767'
```

Chương trình hoàn chỉnh sau minh họa cho cách dùng **f-string** để tạo khuôn dữ liệu.

fstring.py

```
1. DS =
   [{"Hà", 8.6, "98976659"}, {"Bình", 8.9, "0887655444"}, {"Hoa", 7.89,
   "9887665555"}]
2. print("Họ và tên      Điểm số      Điện thoại")
3. for hoten, diem, dt in DS:
4.     s = f"{hoten:<20} {diem:^6.2f} {dt:>12}"
5.     print(s)
```



Câu hỏi - bài tập

1. Nếu $S = \text{"Việt Nam"}$ thì biểu thức sau thể hiện gì?

```
S[ : : -1]
```

a) "Nam Việt"

b) "teejiV maN"

c) "maN tẹiV"

d) Chương trình báo lỗi.

2. Đoạn chương trình sau sẽ in ra kết quả gì?

```
msg = "Hà Nội Đà Nẵng Cần Thơ"
for w in msg.split():
    print(w[0], end = " ")
```

a) Hà Nội Đà Nẵng Cần Thơ

b) HàNỘIĐàNẵgCầnThơ

c) HNĐNCT

d) H N Đ N C T

3. Kết quả các lệnh sau là gì?

```
>>>s = "abcdefg"
>>>s[-1 : 0 : -1]
```

a) "" (xâu rỗng)

b) "gfedcba"

c) "abcdefg"

d) "gfedcb"

4. Kết quả các lệnh sau là gì?

```
>>>s = "abcdefg"
>>>s[-1 : : -1]
```

a) "" (xâu rỗng)

b) "gfedcba"

c) "abcdefg"

d) "gfedcb"

5. Cho trước xâu ký tự s . Viết đoạn chương trình tách các ký tự là số của xâu trên và chuyển vào một dãy A .

Ví dụ nếu $s = \text{"01-tháng-5-năm 2020"}$ thì kết quả dãy A sẽ là:

$A = [0, 1, 5, 2, 0, 2, 0]$.

6. Giả sử s là một xâu ký tự bất kỳ. Khi đó lệnh sau:

```
>>>len(s.split())
```

Có ý nghĩa gì?

7. Hãy giải thích ý nghĩa đoạn chương trình sau:

```
>>>S = " Bui Viet Ha"  
>>>S = S.strip()  
>>>S1 = S[ : S.find(" ")]
```

Kết quả của ký tự S1 thu được là gì?

8. Đoạn chương trình sau cho kết quả là gì?

```
1. S = "Bùi Việt Hà"  
2. A = S.split()  
3. A = A[ : : -1]  
4. S1 = " ".join(A)  
5. S2 = S1[0 : S1.find(" ")]  
6. print(S2)
```

9. Lệnh sau cho kết quả như thế nào?

```
>>> ord("ABC")
```

a) 65

b) h41

c) 1000001

d) Chương trình báo lỗi.

10. Kết quả của chương trình sau sẽ in như thế nào?

```
1. s = "ab2vc4f7h8"  
2. kq = ""  
3. for ch in s:  
4.     if not (ch >= '0' and ch <= '9'):  
5.         kq = kq + ch  
6. print(kq)
```

a) ab2vc4f7h8

b) abvcfh

c) 2478

d) Chương trình báo lỗi.

11. Viết thủ tục tính số các từ đơn của một đoạn văn bản - chuỗi ký tự bất kỳ. Ví dụ với chuỗi: s = " Hà Nội thủ đô của Việt Nam ", thì kết quả trả lại là 7.

12. Viết thủ tục **fullstrip(s)** có chức năng chỉnh sửa dấu cách của chuỗi ký tự bất kỳ s, xóa dấu cách đầu, cuối và xóa các dấu cách thừa bên trong chuỗi s.

Ví dụ với s = " Hà Nội thủ đô của Việt Nam " thì kết quả phải là:

fullstrip(s) = "Hà Nội thủ đô của Việt Nam".

13. Viết hàm số **TachTen(Hoten)** có chức năng tách phần tên tiếng Việt của một họ tên đầy đủ Hoten. Ví dụ Hoten = "Nguyễn Quang Bình" thì hàm sẽ trả lại "Bình".

14. Cho trước (hoặc nhập từ bàn phím) chuỗi ký tự Str. Hãy đếm số các ký tự là số trong chuỗi trên.

15. Giải thích ý nghĩa của chương trình sau:

```
1. sin = "Chiến khu ATK Việt Bắc"
2. sout = ""
3. for ch in sin:
4.     sout = sout + chr(ord(ch) + 2)
5. print(sout)
```

Kết quả chương trình in ra là: Ejkkp"mjw"CVM"Xkiv"Dăc

16. Giải thích ý nghĩa của chương trình sau.

```
1. sin = '''Ejkkp"mjw"CVM"Xkiv"Dăc'''
2. sout = ""
3. for ch in sin:
4.     sout = sout + chr(ord(ch) - 2)
5. print(sout)
```

Kết quả đầu ra của chương trình trên là gì? So sánh với chương trình bài 15.

17. Hãy giải thích kết quả chương trình sau:

```
1. sin = "Chiến khu ATK Việt Bắc"
2. sout = ""
3. for ch in sin:
4.     if ch == " ":
5.         sout = sout + " "
6.     else:
7.         sout = sout + chr(ord(ch) + 2)
8. print(sout)
```

18. Viết hàm số **Component(S,k)** có ý nghĩa như sau:

S: Xâu ký tự đầu vào.

k: Số tự nhiên.

Kết quả hàm số trả lại thành phần, từ đơn thứ k của xâu S. Nếu số k nằm ngoài phạm vi cho phép thì hàm trả lại xâu rỗng.

Ví dụ:

S = "Hà Nội là thủ đô của Việt Nam".

Component(S, 2) = "Nội".

Component(S, 8) = "Nam".

Component(S, 9) = "".

19. Cho trước (hoặc nhập từ bàn phím) xâu ký tự Str. Hãy đếm số các ký tự là chữ cái tiếng Anh (kể cả chữ hoa, chữ thường) trong xâu Str.

20. Cho trước (hoặc nhập từ bàn phím) xâu ký tự Str, đếm số các ký tự không là số, không là chữ cái tiếng Anh trong xâu Str.

21 Cho trước 2 xâu Str1, Str2. Viết chương trình:

a) Kiểm tra xem xâu Str2 có nằm trong Str1 hay không?

b) Nếu có thì Str2 được xuất hiện (có thể chồng lên nhau) trong Str1 bao nhiêu lần.

c) Tương tự b) nhưng chỉ tính số lần xuất hiện không chồng lên nhau.

22. Cho trước (hoặc nhập từ bàn phím) xâu ký tự Str, in ra xâu thu được từ Str bằng cách viết theo thứ tự ngược lại.
23. Cho trước 2 xâu Str1, Str2, viết chương trình trộn 2 xâu trên lại, in ra kết quả. Việc trộn xâu theo nguyên tắc sau:
- Lần lượt từ trái sang phải, viết các ký tự của Str1, sau đó đến Str2.
 - Nếu một trong 2 xâu kết thúc thì chỉ viết ra các ký tự của xâu còn lại.
24. Cho trước (hoặc nhập từ bàn phím) xâu ký tự Str. Hãy xóa đi tất cả các ký tự của Str không là số, xâu còn lại sẽ là số. Hãy in ra xâu kết quả này và thông báo nó là nguyên tố hay không.
25. Cho trước (hoặc nhập từ bàn phím) xâu ký tự Str. Hãy xóa đi tất cả các ký tự không là chữ cái tiếng Anh của xâu này. In ra xâu kết quả.
26. Cho trước xâu ký tự nhị phân bao gồm toàn các ký tự 0 và 1. Viết chương trình chuyển đổi số nhị phân này sang số thập phân.
27. Cho trước xâu Str. Kiểm tra xem Str có phải là xâu nhị phân (toàn 0, 1) hay không. Nếu có thì chuyển xâu này sang số thập phân. In kết quả số thập phân trên màn hình.
28. Cho trước số tự nhiên n được viết dưới dạng hệ thập phân. Viết chương trình chuyển số n từ hệ đếm thập phân sang nhị phân. Kết quả là xâu nhị phân.
29. Cho trước xâu Str. Kiểm tra xem Str có phải là xâu được viết trong hệ hex (0, 1, ..., 9, A, B, C, D, E, F) hay không. Nếu có thì chuyển xâu này sang số thập phân. In kết quả số thập phân trên màn hình.
30. Cho trước (hoặc nhập từ bàn phím) xâu ký tự Str. Hãy đảo các số là chữ số sang bên trái (giữ đúng thứ tự) và in ra kết quả. Ví dụ:
- Input: abc9k00ty8n0.
Output: 90080abcktytn.
31. Cho trước 2 xâu Str và St, số $k < \text{length}(\text{Str})$. Viết chương trình chèn xâu St vào Str tại vị trí ký tự thứ k. In kết quả ra màn hình.
32. Nhập từ bàn phím một họ tên người Việt Nam đầy đủ, ví dụ: Bùi Việt Hà. Viết chương trình tách và in ra họ, đệm, tên của người này. Ví dụ:
- Input: Bùi Việt Hà. Output: họ - Bùi, đệm - Việt, tên - Hà.
Input: Nguyễn Khang. Output: họ - Nguyễn, đệm không có, tên - Khang.
Input: Bạch Thị Thúy Hoa. Output: họ - Bạch, đệm - Thị Thúy, tên - Hoa.
33. Nhập một họ tên đầy đủ, viết chương trình sửa, xóa đi các dấu cách thừa của họ tên này. Chú ý dấu cách thừa có thể ở giữa các họ, đệm, tên hoặc ở đầu và cuối xâu.
34. Cho trước xâu Str là một đoạn văn bản hoàn chỉnh (có thể nhiều dòng). Nhập một từ đơn, hãy tìm xem trong văn bản Str chứa bao nhiêu từ đơn này.
35. Nhập một xâu bất kỳ từ bàn phím. Viết chương trình kiểm tra nếu thấy có 2 ký tự liền nhau và giống nhau thì xóa đi một ký tự. Ví dụ nhập vào là aabbbccc thì đưa ra kết quả abc.
36. Viết chương trình nhập lần lượt thông tin học sinh trong lớp theo các thông số sau:

- Nhập họ tên học sinh.
- Nhập điểm số môn Toán (số thập phân với 1 chữ số sau dấu phẩy).
- Nhập điểm số môn Văn (số thập phân với 1 chữ số sau dấu phẩy).

Việc nhập dữ liệu kết thúc khi người dùng nhập một xâu rỗng.

Sau đó in ra màn hình học tên học sinh, điểm số môn Toán, Văn theo mẫu sau:

Họ tên	Toán	Văn
Nguyễn Văn Quang	8.5	9.1
Bùi Quang Bình	7.0	8.6
Trần Thu Hà	10	9.8
Lê Quang My	9.7	5.6

Yêu cầu thể hiện ra như sau:

- Họ tên học sinh được căn trái, chiếm khoảng rộng 30 ký tự.
- Điểm Toán, chiếm khoảng rộng 5, căn phải.
- Điểm Văn, chiếm khoảng rộng 5, căn phải.

37. Xâu ký tự có độ dài n được gọi là đối xứng nếu $s[k] = s[n-k-1]$ với $k = 0, 1, 2, \dots, n-1$. Viết hàm số **isSymmetry(s)** kiểm tra xem xâu S có phải là đối xứng hay không.

38. Viết chương trình nhập từ bàn phím một danh sách tên học sinh trong lớp. Cần nhập họ đệm và tên đầy đủ. Ví dụ có thể nhập một danh sách như sau:

Nguyễn Văn Quang
Bùi Quang Thành
Trần Văn An
Lê Quang Bình

Chương trình sẽ sắp xếp lại danh sách học sinh theo thứ tự từ điển nhưng theo nguyên tắc sắp xếp tên trước họ đệm sau. Với bộ dữ liệu trên, kết quả sắp xếp sẽ phải như sau:

Trần Văn An
Lê Quang Bình
Nguyễn Văn Quang
Bùi Quang Thành

39. Cho trước (hoặc nhập từ bàn phím) xâu ký tự Str . Hãy tìm một xâu con có độ dài cực đại và bao gồm các phần tử giống nhau của xâu trên. Ví dụ nếu xâu vào là $aabbcc$ thì output là bbb .

40. Cho trước (hoặc nhập từ bàn phím) 2 xâu ký tự $Str1$ và $Str2$. Hãy tìm ra xâu con chung có độ dài lớn nhất của 2 xâu trên.

Chương 10. Đọc và ghi tệp

Mục tiêu:

- Nhận biết kiểu file.
- Các lệnh đọc và ghi file dạng plain text.



Nội dung chính

Trong bài học này chúng ta sẽ làm quen với đối tượng tệp (file) và các lệnh làm việc với file. Đối tượng làm việc của chúng ta trong bài học này là các Text File, tức là các tệp chứa ký tự và văn bản. Mặc định Python sẽ có các phương thức để làm việc với các tệp chỉ chứa các ký tự lấy từ bảng mã ASCII, do vậy tất cả các tệp văn bản trong bài học này đều hạn chế với dữ liệu trong bảng ASCII. Chú ý rằng các tệp này không thể chứa các từ, chữ tiếng Việt vì chữ cái tiếng Việt vượt ra khỏi bảng ASCII.

Các tệp văn bản thường có phần mở rộng là **txt** hay **dat**.

Chúng ta sẽ làm việc với các tệp văn bản mẫu, ví dụ một tệp như vậy có nội dung sau.

Inp.dat

```
This is plain text data file
1 8 10 15 12
12.4 17.99 3.27 -14.66 1.2
```

```
A B C a b c
```

```
This is the last line
```

Lệnh mở file thường dùng là lệnh `open()`. Lệnh `open()` có 2 tham số:

`open(<Tên File>, mode)`

trong đó <Tên File> là tên tệp văn bản cần mở, mode sẽ có dạng hoặc là "r" - chỉ đọc và "w" - mở để ghi.

Chúng ta tạo một tệp chương trình như sau và đặt tệp Inp.dat cùng thư mục với tệp này. Chương trình sẽ tạo đối tượng file thông qua lệnh `open()`, sau đó dùng phương thức `read()` để đưa toàn bộ nội dung của tệp vào biến nhớ `content`, sau đó in ra nội dung này.

Docfile.py

```
1. # chương trình đọc file
2. file = open("Inp.dat", "r")
3. content = file.read()
4. print(content)
```

Kết quả chạy của chương trình trên như sau. Chúng ta thấy toàn bộ thông tin của tệp Inp.dat sẽ được in ra đúng như nội dung bên trong tệp.

```
This is plain text data file
1 8 10 15 12
12.4 17.99 3.27 -14.66 1.2
```

```
A B C a b c
```

```
This is the last line
```

Nếu cần mở tệp trong một thư mục khác, hoặc mở tệp ngay từ Python Shell thì cần thiết lập thư mục làm việc hệ thống dẫn đến thư mục chứa tệp văn bản này.

Cách thiết lập thư mục làm việc hiện thời của Python như sau.

```
>>> import os
>>> os.chdir(<Path>)
```

Ví dụ trong trường hợp của chúng ta, có thể thực hiện lệnh sau.

Chú ý là chỉ cần thực hiện lệnh này một lần.

```
os.chdir("D:\Habv\docx\Habv\Habv_hoc_Python\Sach-hoc-
Python\Python-Coban\Projects\B10")
```

Bây giờ chúng ta có thể thực hiện các lệnh làm việc với đối tượng file ngay trên màn hình tương tác của Python. Đối tượng làm việc sẽ thuộc kiểu file.

```
>>> file = open("Inp.dat", "r")
>>> s = file.read()
>>> type(file)
<class '_io.TextIOWrapper'>
>>> type(s)
<class 'str'>
>>> s
'This is plain text data file\n1 8 10 15 12 \n12.4 17.99 3.27 -
14.66 1.2\n\nA B C a b c \nThis is the last line'
>>>
```

Đối tượng file có kiểu với tên hơi lạ là **_io.TextIOWrapper**.

Nội dung của tệp sau lệnh `open()` sẽ là một chuỗi ký tự. Chú ý các dãy `\n` được chèn vào chuỗi này.

Danh sách các thuộc tính và phương thức của lớp này như sau.

```
>>> dir(file)
['_CHUNK_SIZE', '_class_', '_del_', '_delattr_',
'_dict_', '_dir_', '_doc_', '_enter_', '_eq_',
'_exit_', '_format_', '_ge_', '_getattr_',
'_getstate_', '_gt_', '_hash_', '_init_',
'_init_subclass_', '_iter_', '_le_', '_lt_', '_ne_',
'_new_', '_next_', '_reduce_', '_reduce_ex_', '_repr_',
'_setattr_', '_sizeof_', '_str_', '_subclasshook_',
'_checkClosed', '_checkReadable', '_checkSeekable',
'_checkWritable', '_finalizing', 'buffer', 'close', 'closed',
'detach', 'encoding', 'errors', 'fileno', 'flush', 'isatty',
'line_buffering', 'mode', 'name', 'newlines', 'read', 'readable',
'readline', 'readlines', 'reconfigure', 'seek', 'seekable',
'tell', 'truncate', 'writable', 'write', 'write_through',
'writelines']
>>>
```

Các phương thức chính với đối tượng file in đậm trong danh sách trên sẽ được mô tả trong các phần tiếp theo của bài học.

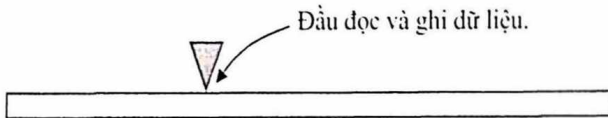
1. Các lệnh đọc thông tin từ tệp

1.1. Các lệnh `open()`, `read()`, `close()`, `tell()`

Lệnh `open()` sẽ tạo ra một đối tượng mới dạng tệp và chuẩn bị sẵn sàng cho các công việc đọc và ghi.

Việc đọc hay ghi dữ liệu từ tệp sẽ thông qua một đầu đọc (file head). Khi đọc hoặc ghi dữ liệu sẽ đọc hay ghi trực tiếp từ vị trí này. Khi đọc xong (hay ghi xong), đầu đọc sẽ chuyển dịch về cuối tệp đúng một khoảng bằng lượng thông tin (tính theo byte) đã đọc hoặc đã ghi.

Mô hình đầu đọc tệp như hình sau.



Lệnh `open(<tên file>, "r")` sẽ mở tệp ra sẵn sàng đọc. Đầu đọc sẽ chuyển về đầu tệp, vị trí gốc là 0.

Lệnh `open(<tên file>, "w")` sẽ mở tệp, xóa mọi thông tin đã có trong tệp và đưa đầu ghi về vị trí đầu tệp. Nếu tệp chưa có thì lệnh này sẽ tạo ra một tệp văn bản rỗng và sẵn sàng cho việc ghi dữ liệu.

Quan sát ví dụ sau với 2 lần đọc dữ liệu bằng lệnh `read()`.

```
>>> f = open("Inp.dat", "r")
>>> content = f.read()
>>> print(content)
This is plain text data file
1 8 10 15 12
12.4 17.99 3.27 -14.66 1.2

A B C a b c
This is the last line
>>> content = f.read()
>>> content
''
>>>
```

Lần đọc đầu tiên, `content` = toàn bộ nội dung của tệp.

Lần đọc tiếp theo, đầu đọc đã ở cuối tệp nên `content` = rỗng.

Sau lần gọi đọc đầu tiên, toàn bộ dữ liệu từ Text File đã được đọc vào biến nhớ `content`. Biến nhớ này là kiểu dữ liệu str. Khi đọc lần thứ 2, dữ liệu đọc vào là rỗng. Điều này để hiểu khi biết cơ chế đầu đọc dữ liệu của Python. Lệnh `read()` luôn thực hiện việc đọc dữ liệu từ vị trí đầu đọc trở đi. Do lần đầu đã đọc hết tệp, nên lần 2, vì đầu đọc đã ở cuối tệp nên không có dữ liệu nữa để đọc. Làm thế nào để biết được đầu đọc tệp đang ở vị trí nào? Hàm `tell()` sẽ giúp chúng ta hiểu rõ hơn điều này.

Hàm `tell()` luôn chỉ ra vị trí hiện thời của đầu đọc tệp tính từ vị trí đầu tệp. Tại vị trí gốc, giá trị này = 0. Trong suốt quá trình đọc dữ liệu từ tệp, khi đầu đọc di chuyển, hàm `tell()` sẽ chỉ ra vị trí hiện thời chính bằng số lượng byte thông tin đã được đọc.

Ví dụ sau minh họa cho điều trên.

```
>>> f = open("Inp.dat")
>>> f.tell() -----> Bắt đầu mở tệp, đầu đọc chỉ = 0.
0
>>> all = f.read()
>>> f.tell() -----> Đọc xong, đầu đọc chỉ cuối tệp = 110.
110
>>>
```

Khi kết thúc đọc dữ liệu thì cần thực hiện lệnh đóng tệp `close()` để chuyển sang làm việc với các tệp khác.

Chú ý khi đã đóng tệp rồi thì không thể thực hiện bất cứ lệnh nào nữa (như đọc, ghi, tell) mà phải mở lại trước khi thực hiện các lệnh này.

Ví dụ sau mô tả khi đã đóng tệp thì các lệnh như `read()`, `tell()` đều không thể thực hiện được.

```
>>> f.close()
>>> f.tell()
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    f.tell()
ValueError: I/O operation on closed file.
>>> all = f.read()
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    all = f.read()
ValueError: I/O operation on closed file.
>>>
```

1.2. Lệnh `readline()`

Khác với lệnh `read()` là đọc toàn bộ nội dung của tệp, lệnh `readline()` sẽ đọc theo từng dòng của tệp. Lệnh này khá thú vị và hay được dùng trên thực tế.

Quan sát ví dụ sau, chúng ta luôn chèn các lệnh `tell()` để kiểm tra vị trí đầu đọc file đang ở đâu.

```
>>> f = open("Inp.dat")
>>> f.tell() -----> Khi mới mở tệp, vị trí đầu đọc ở
0                               đầu file, vị trí = 0.
>>> line = f.readline()
>>> line
'This is plain text data file\n'
>>> f.tell() -----> Đọc xong dòng thứ nhất, vị trí
30                               đầu đọc = 30.
>>> line = f.readline()
>>> line
'1 8 10 15 12 \n'
>>> f.tell() -----> Đọc xong dòng thứ 2, vị trí đầu
45                               đọc = 45.
>>>
```

Các ví dụ sau là những ứng dụng thực tế của lệnh `readline()` thường dùng để đọc dữ liệu từ file.

Ví dụ 1: Đọc dữ liệu từ tệp văn bản gồm một dòng chứa 2 số tự nhiên m, n . Giả sử tệp dữ liệu là "inp1.dat".

DocFile1.py

```
1. file = open("Inp1.dat")
2. line = file.readline()
3. A = line.split()
4. m = int(A[0])
5. n = int(A[1])
6. print("Các số đã nhập là:", m, n)
```

Ví dụ 2: Đọc dữ liệu từ tệp văn bản có 3 dòng, mỗi dòng chứa một số tự nhiên lần lượt ký hiệu là m, n, p . Giả sử tệp dữ liệu là "inp2.dat".

DocFile2.py

```
1. file = open("Inp2.dat")
2. line = file.readline()
3. m = int(line)
4. line = file.readline()
5. n = int(line)
6. line = file.readline()
7. p = int(line)
8. print("Các số đã nhập là:", m, n, p)
```

1.3. Lệnh `readlines()`

Lệnh `readline()` chúng ta đã biết trong phần trên là đọc từng dòng của tệp. Python còn có một lệnh nữa, gần tương tự về tên gọi, nhưng ý nghĩa khác hẳn, đó là lệnh `readlines()`. Lệnh này sẽ đọc toàn bộ nội dung tệp từ vị trí đầu đọc hiện thời đến cuối tệp và kết quả trả lại một list các dòng của tệp.

Trong ví dụ sau, kết quả đọc tệp "Inp.dat" từ đầu sẽ đưa vào danh sách `a`, sau đó in ra màn hình.

```
>>> file = open("Inp.dat")
>>> a = file.readlines()
>>> a
['This is plain text data file\n', '1 8 10 15 12 \n', '12.4 17.99\n', '3.27 -14.66 1.2\n', '\n', 'A B C a b c \n', 'This is the last line']
>>> for line in a:
    print(line, end = "")
This is plain text data file
1 8 10 15 12
12.4 17.99 3.27 -14.66 1.2

A B C a b c
This is the last line
>>>
```

Chú ý:

Cuối từng phần tử của dãy `a` đã có ký tự xuống dòng "`\n`" nên cần đưa tham số lựa chọn `end=""` để khi in ra không bị xuống dòng 2 lần,

Ví dụ sau lệnh đọc `readlines()` được thực hiện khi đầu đọc đã ở giữa tệp. Lệnh sẽ đọc các dòng còn lại và đưa vào một danh sách.

```
1. >>> f = open("Inp.dat")
2. >>> line = f.readline()           # đọc dòng 1
3. >>> line
4. 'This is plain text data file\n'
5. >>> line = f.readline()           # đọc dòng 2
6. >>> line
7. '1 8 10 15 12 \n'
8. >>> f.tell()                       # đang ở đầu dòng thứ 3
9. 45
10. >>> a = f.readlines()              # đọc tiếp từ dòng thứ 3
11. >>> a
12. ['12.4 17.99 3.27 -14.66 1.2\n', '\n', 'A B C a b c \n', 'This
    is the last line']
13. >>> f.tell()                       # đã đến cuối tệp
14. 110
15. >>>
```

1.4. Đọc file như kiểu dữ liệu tuần tự

Chúng ta còn một cách nữa để đọc thông tin từ tệp. Đó là cách coi file như kiểu dữ liệu tuần tự bao gồm các dòng và có thể dùng lệnh **for** để đọc lần lượt các dòng.

Xem ví dụ sau để hiểu hơn cách đọc này.

```
>>> f = open("Inp.dat")
>>> for line in f:                     →  Biến nhớ line lần lượt lấy
    print(line, end = "")              thông tin là các dòng của f.

This is plain text data file
1 8 10 15 12
12.4 17.99 3.27 -14.66 1.2

A B C a b c
This is the last line
>>>
```

2. Các lệnh ghi thông tin ra tệp

Công việc ghi dữ liệu ra file có thể coi là các thao tác ngược với lệnh đọc. Nếu như chúng ta có các lệnh đọc `read()`, `readline()` và `readlines()` thì chắc ai trong chúng ta cũng nghĩ rằng sẽ có các lệnh ghi dữ liệu tương tự, ví dụ như có thể có các lệnh `write()`, `writeline()` và `writelines()`.

Tuy nhiên thực tế không như vậy. Các lệnh ghi dữ liệu của Python bao gồm: `write()`, `writelines()` và `print()`. Các lệnh này có ý nghĩa khác với các lệnh đọc tương ứng. Chúng ta sẽ cùng tìm hiểu các lệnh này.

Muốn ghi dữ liệu ra tệp chúng ta cần tạo đối tượng tệp, ví dụ như sau:

```
file = open(<tên tệp>, "w")
```

Lệnh này sẽ tạo ra đối tượng file sẵn sàng cho việc ghi dữ liệu. Nếu tệp chưa có trên máy tính, Python sẽ tạo mới tệp này. Nếu tệp đã có thì Python sẽ xóa hết dữ liệu và sẵn sàng cho việc ghi.

Tương tự như việc đọc, lệnh ghi cũng sẽ thực hiện qua đầu đọc/ghi của tệp. Ban đầu, đầu đọc ở vị trí 0 (đầu tệp), khi thực hiện lệnh ghi, đầu đọc sẽ di chuyển theo đúng dung lượng thông tin đã ghi, hàm `tell()` sẽ luôn trả lại vị trí của đầu đọc hiện thời.

2.1. Ghi dữ liệu bằng lệnh `write()`

Lệnh `write()` có một tham số duy nhất là một chuỗi ký tự. Lệnh này sẽ ghi chuỗi ký tự này ra file đang mở tại vị trí đầu đọc hiện thời.

Quan sát ví dụ sau:

GhiFile1.py

```
1. fileout = open("Out1.dat", "w")
2. msg1 = "The first line."
3. msg2 = "12 6.7 3.14"
4. msg3 = "The last line."
5. fileout.write(msg1)
6. fileout.write(msg2)
7. fileout.write(msg3)
8. fileout.close()
```

Chương trình lần lượt ghi ra file 3 chuỗi ký tự `msg1`, `msg2`, `msg3` bằng lệnh `write()`.

Kết quả tệp được ghi ra sẽ như sau.

Out1.dat

```
The first line.12 6.7 3.14The last line.
```

Trong tệp kết quả, 3 chuỗi trên sẽ nối liền nhau mà không có xuống dòng. Như vậy lệnh `write()` sẽ ghi ra tệp chuỗi ký tự đúng trong tham số của mình và không chèn thêm ký tự xuống dòng `"\n"`.

Muốn sau mỗi lệnh `write()` xuống dòng thì cần chèn dãy điều khiển `"\n"` vào cuối của mỗi chuỗi ký tự của lệnh `write()`. Xem chương trình sau.

GhiFile2.py

```
1. fileout = open("Out2.dat", "w")
2. msg1 = "The first line.\n"
3. msg2 = "12 6.7 3.14\n"
4. msg3 = "The last line.\n"
5. fileout.write(msg1)
6. fileout.write(msg2)
7. fileout.write(msg3)
8. fileout.close()
```

Các chuỗi ký tự `msg1`, `msg2`, `msg3` đều có ký tự `"\n"` ở cuối.

Kết quả tệp đầu ra có khuôn dạng 3 dòng như sau.

Out2.dat

```
The first line.
12 6.7 3.14
The last line.
```

2.2. Ghi ra tệp bằng lệnh print()

Lệnh print() mà chúng ta đã biết cũng có thể thực hiện việc “in” thông tin ra tệp thay thế in ra màn hình (hay thiết bị đầu ra chuẩn). Muốn ghi thông tin ra File thì lệnh print() sẽ cần thêm một tham số nữa, tham số file = <>.

Cú pháp của lệnh print() ghi ra tệp như sau.

```
print(<xâu ký tự>, file = <đối tượng File>)
```

Lệnh trên mặc định sẽ ghi ra tệp kết quả <xâu ký tự> và cuối xâu sẽ chèn ký tự xuống dòng “\n”. Do vậy lệnh trên sẽ tương đương với lệnh write() như sau.

```
<đối tượng File>.write(<xâu ký tự> + “\n”)
```

Muốn có lệnh print() ghi ra file hoàn toàn tương thích với lệnh write() chúng ta có thể thực hiện lệnh sau, bổ sung thêm tham số end = “”.

```
print(<xâu ký tự>, file = <đối tượng File >, end = “”)
```

Lệnh này sẽ tương đương với lệnh write() như sau.

```
<đối tượng File>.write(<xâu ký tự>)
```

Chúng ta viết lại ví dụ phía trên bằng lệnh print() như sau.

GhiFilePrint1.py

```
fileout = open("Out1.dat", "w")
msg1 = "The first line."
msg2 = "12 6.7 3.14"
msg3 = "The last line."
print(msg1, file = fileout)
print(msg2, file = fileout)
print(msg3, file = fileout)
fileout.close()
```

Kết quả ghi ra tệp Out1.dat như sau.

Out1.dat

```
The first line.
12 6.7 3.14
The last line.
```

Chú ý quan trọng:

Lệnh print() khi ghi ra file hỗ trợ cho nhiều tham số với nhiều kiểu dữ liệu khác nhau. Khi ghi ra file, các đối tượng dữ liệu này sẽ tự động chuyển sang kiểu xâu ký tự. Tính năng này sẽ làm cho phương pháp ghi ra file bằng lệnh print() rất hay được sử dụng.

Cú pháp tổng thể của lệnh print() ghi ra tệp văn bản như sau.

```
print(<object1>, <Object2>, ... <objectN>, file = <đối tượng File>)
```

Sau đây là ví dụ.

GhiFilePrint2.py

```
1. fileout = open("Out2.dat", "w")
2. m = 10
3. n = 23
4. p = 3.14
5. print("3 numbers m, n, p are:", file = fileout)
6. print(m, n, p, file = fileout)
7. fileout.close()
```

Kết quả ra tệp Out2.dat như sau:

```
3 numbers m, n, p are:
10 23 3.14
```

Như vậy chúng ta thấy lệnh `print()` có thể dùng rất tốt cho công việc ghi dữ liệu ra file.

2.3. Lệnh `writelines()`

Lệnh `writelines()` là một lựa chọn khác của việc ghi thông tin ra file. Lệnh này có một tham số duy nhất là một danh sách (list) các xâu ký tự.

Khi thực hiện lệnh này, các phần tử của dãy tham số (là các xâu ký tự) sẽ lần lượt được ghi vào file tương tự như lệnh `write()` mà chúng ta đã biết.

Cú pháp của lệnh này như sau.

```
<file>.writelines(<danh sách các xâu ký tự>)
```

Chúng ta cùng viết lại 2 chương trình `GhiFile1.py`, `GhiFile2.py` nhưng theo lệnh `writelines()` mới. Các tệp mới sẽ có tên `GhiFile3.py`, `GhiFile4.py`.

GhiFile3.py

```
1. fileout = open("Out3.dat", "w")
2. msg1 = "The first line."
3. msg2 = "12 6.7 3.14"
4. msg3 = "The last line."
5. A = [msg1, msg2, msg3]
6. fileout.writelines(A)
7. fileout.close()
```

Các xâu ký tự này không có "\n" ở cuối xâu.

Chú ý các xâu ký tự `msg1`, `msg2`, `msg3` không có dấu xuống dòng ở cuối.

Kết quả ra như sau:

Out3.dat

```
The first line.12 6.7 3.14The last line.
```

Chương trình sau giống hệt chương trình trên, điểm khác biệt chỉ ở chỗ các xâu ký tự trong danh sách `A` đều có chèn "\n" vào cuối xâu.

```

1. fileout = open("Out4.dat", "w")
2. msg1 = "The first line.\n"
3. msg2 = "12 6.7 3.14\n"
4. msg3 = "The last line.\n"
5. A = [msg1, msg2, msg3]
6. fileout.writelines(A)
7. fileout.close()

```

Các xâu ký tự này đều có "\n" ở cuối xâu.

Kết quả như sau:

Out4.dat

```

The first line.
12 6.7 3.14
The last line.

```

3. Một số ứng dụng đọc và ghi tệp

Sau đây là một số bài toán cụ thể minh họa cho việc vào/ra file của Python.

Bài toán 1. Viết chương trình tính ƯSCLN của 2 số tự nhiên m, n và in kết quả ra tệp. Dữ liệu đầu vào bao gồm nhiều dòng, mỗi dòng là một bộ 2 số tự nhiên. Dữ liệu đầu ra bao gồm nhiều dòng, mỗi dòng là ƯSCLN của 2 số tương ứng của tệp đầu vào. Tên tệp đầu vào là input1.dat, tệp đầu ra là output1.dat. Ví dụ các tệp này như sau.

input1.dat	output1.dat
12 3	3
25 35	5
40 30	10

Chương trình có thể như sau.

CT1.py

```

1. # Chương trình 1, tính Ước số chung lớn nhất 2 số.
2. fin = open("input1.dat")
3. fout = open("output1.dat", "w")
4. def uscln(m, n):
5.     while m != n:
6.         if m < n:
7.             n = n - m
8.         else:
9.             m = m - n
10.    return m
11. for line in fin:
12.    x, y = line.split()
13.    m = int(x)
14.    n = int(y)
15.    print(uscln(m,n), file = fout)
16. fin.close()
17. fout.close()

```

Bài toán 2. Viết chương trình với đầu vào là một dãy các số tự nhiên, mỗi số một hàng. Với mỗi số tự nhiên, chương trình sẽ tính và in ra được dãy các ước số nguyên tố của số này, tính cả các số bị lặp lại. Kết quả in ra là tệp văn bản với các dòng, mỗi dòng là dãy các ước số nguyên tố của số tự nhiên tương ứng trong tệp đã đọc.

input2.dat	output2.dat
10	2 5
24	2 2 2 3
15	3 5

Chương trình có thể như sau.

CT2.py

```

1. # Chương trình 2, khai triển số thành dãy các ước số nguyên tố.
2. fin = open("input2.dat")
3. fout = open("output2.dat", "w")
4. def nguyento(n):
5.     d = []
6.     k = 2
7.     while k <= n:
8.         i, r = divmod(n, k)
9.         if r == 0:
10.            n = i
11.            d.append(k)
12.        else:
13.            k = k+1
14.     return d
15. for line in fin:
16.     n = eval(line)
17.     d = nguyento(n)
18.     for i in d:
19.         print(i, file = fout, end = " ")
20.     print(file = fout)
21. fin.close()
22. fout.close()

```

4. Tổng kết các phương thức đọc và ghi tệp

Bảng sau tổng kết các lệnh và cách đọc, ghi file trong Python.

Phương thức	Cú pháp lệnh	Giải thích
Lệnh mở file	open(<Tên file>, <type>)	type = "r", kiểu đọc file. Mở file để đọc. type = "w", kiểu ghi file. Mở file để ghi.
Lệnh read()	<file>.read()	Đọc toàn bộ nội dung của file vào một string.
Lệnh readline()	<file>.readline()	Đọc một dòng (tiếp theo) từ <file> vào một string.

Phương thức	Cú pháp lệnh	Giải thích
Lệnh <code>readlines()</code>	<code><file>.readlines()</code>	Đọc toàn bộ nội dung của tệp <code><file></code> vào một danh sách (list) mà mỗi phần tử là một string, là 1 dòng của tệp vừa đọc.
Lệnh <code>write()</code>	<code><file>.write(string)</code>	Ghi xâu ký tự string ra file tính từ vị trí đọc hiện thời.
Lệnh <code>writelines()</code>	<code><file>.writelines(List of string)</code>	Ghi một danh sách các xâu ký tự ra file tính từ vị trí đọc hiện thời. Mỗi xâu được ghi ra trên một dòng.
Lệnh <code>print()</code>	<code>print(string, file = <file>, [end = <endstr>])</code>	Ghi thông tin xâu string ra tệp <code><file></code> , tiếp theo là ký tự <code><endstr></code> nếu có, mặc định <code>endstr = ""</code> .
Đọc dữ liệu tuần tự từ file.	<code>for line in <file></code>	Đọc từng dòng dữ liệu từ tệp file và đưa vào biến <code>line</code> .



Câu hỏi - bài tập

Các bài tập sau sẽ cùng sử dụng tệp "DL.txt" có dạng:

```
This is text file
The second line
The last
```

1. Giả sử chúng ta viết đoạn chương trình sau:

```
file = open("DL.txt", "r")
text = file.read()
```

Hỏi biến nhớ **text** sẽ thuộc kiểu dữ liệu nào?

- a) str
- b) list
- c) tuple
- d) file object

2. Giả sử chúng ta viết đoạn chương trình sau:

```
file = open("DL.txt", "r")
text = file.readlines()
```

Hỏi biến nhớ **text** sẽ thuộc kiểu dữ liệu nào?

- a) str
- b) list
- c) tuple
- d) file object.

3. Chương trình sau in ra kết quả như thế nào?

```
File = open("DL.txt", "r")
for line in File:
    print(line)
File.close()
```

a)

"This is text file"

"The second line"

"The last"

b)

This is text file

The second line

The last

c)

"This is text file\n The second line\n The last\n"

4. Viết chương trình nhập dữ liệu từ tệp text file có dạng như sau:

- Dòng đầu tiên là số tự nhiên n.

- Dòng thứ 2 là n số được viết cách nhau bởi dấu cách.

Ví dụ một tệp như vậy.

```
5
19 -2 9.8 7 -10
```

Chương trình cần đọc dữ liệu đưa vào biến nhớ n và list A, sau đó in n và A ra màn hình.

5. Viết chương trình thực hiện công việc ngược lại so với bài tập trên.

Giả sử cho trước số tự nhiên n và dãy số A bao gồm n số hạng bất kỳ.

Viết chương trình ghi dữ liệu ra một text file có dạng sau:

- Dòng đầu tiên là số tự nhiên n.

- Dòng thứ 2 là các số của dãy A được ghi cách nhau bởi dấu cách.

Ví dụ ghi ra tệp như sau:

```
5
19 -2 9.8 7 -10
```

6. Viết chương trình trên Python thực hiện công việc sau:

- Đọc dữ liệu từ tệp dạng plain text **fin.dat** và đưa dữ liệu ra tệp văn bản **fout.dat**.

- Tệp **fin.dat** có cấu trúc như sau:

+ Dòng đầu tiên là số tự nhiên n.

+ n dòng tiếp theo, mỗi dòng là một dãy các số nguyên hoặc thực, cách nhau bởi dấu cách. Số lượng các phần tử của mỗi dãy có thể không bằng nhau.

- Tệp **fout.dat** có cấu trúc như sau:

- + Dòng đầu tiên ghi số S là tổng của tất cả các số trong các dãy số từ tệp **fin.dat**.
- + n dòng tiếp theo, dòng thứ i ghi số S_i là tổng các số của dãy số tương ứng từ tệp **fin.dat**.

Ví dụ:

fin.dat
3
-1 10 5 4 10
15 -11 80 -10
3 4 7 -8

fout.dat
108
28
74
6

7. Cho trước dãy (list) A bao gồm các số bất kỳ. Viết chương trình ghi ra tệp văn bản "DL.dat" có dạng là nhiều dòng, các dòng là các số lấy từ A.

Ví dụ nếu dãy A có dạng:

$A = [0, 0.3, 5.7, 19, -1.3, 15.56, 7.0]$

thì tệp "DL.dat" sẽ có dạng:

```
0
0.3
5.7
19
-1.3
15.56
7.0
```

8. Bài tập này là ngược lại bài tập 7. Cho trước tệp văn bản "DL.dat" có dạng n dòng, mỗi dòng là một số. Yêu cầu đọc dữ liệu từ tệp này và đưa vào dãy (list) A bao gồm n phần tử vừa đọc từ tệp này.

9. Cho trước ma trận A dạng $m \times n$ (m hàng, n cột) được mô tả bởi danh sách như sau:

$A = [[a_{11}, a_{12}, \dots, a_{1n}], [a_{21}, a_{22}, \dots, a_{2n}], \dots, [a_{m1}, a_{m2}, \dots, a_{mn}]]$

Viết chương trình ghi dữ liệu ma trận A ra tệp văn bản dưới dạng sau:

- Dòng đầu tiên ghi 2 số m, n cách nhau bởi dấu cách.

- m hàng tiếp theo, mỗi hàng ghi dãy các số của hàng tương ứng của ma trận A, các số cách nhau bởi dấu cách.

Tệp dữ liệu phải có dạng như sau:

```
m n
a11 a12 ... a1n
a21 a22 ... a2n
.....
am1 am2 ... amn
```

10. Bài tập này ngược lại với bài tập 9. Giả sử cho tệp văn bản có dạng:

m n

a₁₁ a₁₂ ... a_{1n}

a₂₁ a₂₂ ... a_{2n}

.....

a_{m1} a_{m2} ... a_{mn}

- Dòng đầu tiên ghi 2 số tự nhiên m, n cách nhau bởi dấu cách.

- m dòng tiếp theo, mỗi dòng có n số, các số cách nhau bởi dấu cách.

Cần viết chương trình đọc dữ liệu từ tệp văn bản trên và đưa vào dãy (list) A có dạng:

$A = [[a_{11}, a_{12}, \dots, a_{1n}], [a_{21}, a_{22}, \dots, a_{2n}], \dots, [a_{m1}, a_{m2}, \dots, a_{mn}]]$

11. Viết chương trình đọc dữ liệu từ tệp Input.dat bao gồm n số tự nhiên, mỗi số trên một dòng. Đầu ra là tệp văn bản Output.dat có dạng n dòng, mỗi dòng sẽ ghi số 0 hoặc 1 nếu số tương ứng của tệp Input.dat là hợp số hay nguyên tố.

12. Viết chương trình đọc dữ liệu từ tệp Inp.dat có khuôn dạng bao gồm n số tự nhiên được viết trên một dòng, cách nhau bởi dấu cách. Đầu ra là tệp Out.dat bao gồm n số trên một dòng nhưng các số này đã được sắp xếp theo thứ tự tăng dần.

13. Cho trước dãy số $a_1, a_2, \dots, a_n$ được đọc từ tệp Inp.dat có khuôn dạng bao gồm n số tự nhiên được viết trên một dòng, cách nhau bởi dấu cách. Phần tử a_k được gọi là *cực trị* nếu xảy ra một trong 2 điều sau: $a_{k-1} < a_k > a_{k+1}$ hoặc $a_{k-1} > a_k < a_{k+1}$. Viết chương trình in ra tệp Out.dat gồm 2 dòng:

- Dòng 1 ghi số lượng các điểm cực trị của dãy trên.

- Dòng 2 ghi ra lần lượt tất cả các phần tử cực trị của dãy trên.

14. Viết chương trình đọc tệp dữ liệu Inp.dat bao gồm n dòng, mỗi dòng là một số tự nhiên. Chương trình sẽ đưa kết quả ra tệp Out.dat bao gồm n dòng, mỗi dòng sẽ ghi ra lần lượt các ước số nguyên tố khác nhau của số tương ứng từ tệp Inp.dat. Ví dụ như sau:

Inp.dat	Out.dat
20	2 5
25	5
30	2 3 5

15. Tìm hiểu các phương thức của đối tượng kiểu file và viết chương trình sau.

Giả sử có n tệp văn bản với các tên tệp lưu trong một dãy (list). Các tệp này đều có phần mở rộng là *.dat. Mỗi tệp này đều lưu 1 dòng bao gồm dãy các số nguyên (số lượng phần tử trong mỗi dãy có thể khác nhau). Chương trình sẽ lần lượt đọc các dãy số từ từng tệp này, sau đó tự động ghi kết quả là dãy số tương ứng sau khi đã sắp xếp lại theo thứ tự tăng dần. Các tệp đầu ra tương ứng sẽ có tên lấy từ tệp đầu vào sau khi bổ sung thêm "kq" ở đầu tệp. Ví dụ nếu các tệp ban đầu là "Inp1.dat", "Inp2.dat", "Inp3.dat" thì các tệp kết quả sẽ được tự động đặt tên là "kqInp1.dat", "kqInp2.dat", "kqInp3.dat".

Gợi ý: Sử dụng thuộc tính **name** của đối tượng kiểu file.

Chương 11. Câu lệnh điều kiện

Mục tiêu:

- Tìm hiểu sâu hơn kiểu dữ liệu logic (**bool**).
- Tìm hiểu sâu hơn lệnh **if** và lệnh **while**.



Nội dung chính

Chúng ta đã được làm quen với kiểu dữ liệu bool (logic) trong Chương 3 và các lệnh **if**, **while** có liên quan đến kiểu dữ liệu này. Chúng ta sẽ tìm hiểu sâu hơn về kiểu dữ liệu này và các lệnh tương ứng trong bài học này.

1. Kiểu dữ liệu bool

Kiểu dữ liệu bool là dữ liệu chỉ có 2 trạng thái đúng, sai. Giá trị của kiểu bool trong Python được gọi là True, False.

Dữ liệu kiểu bool thường được định nghĩa bởi các biểu thức so sánh hoặc gán trực tiếp giá trị. Trong ví dụ sau khi gán $a = 1 < 2$ là một phép so sánh trực tiếp thì a sẽ được hiểu ngay là kiểu bool. Giá trị b khi gán $b = \text{True}$ thì tất nhiên là thuộc kiểu dữ liệu này.

```
>>> a = 1 < 2
>>> type(a)
<class 'bool'>
>>> b = True
>>> type(b)
<class 'bool'>
>>>
```

Hàm bool() sẽ tính toán giá trị kiểu bool và trả lại giá trị True (đúng) hay False (sai). Xem các ví dụ sau.

```
>>> a = 1.3 > 3.7
>>> bool(a)
False          # a sai vì thực tế 1.3 < 3.7
>>> bool(False)
False
>>>
```

Trên thực tế không phải các đối tượng dữ liệu nào cũng nhận các giá trị Đúng, Sai tuyệt đối. Có những mệnh đề mà theo nghĩa đời thường chỉ là “gần đúng” hoặc “gần sai”. Python đã tổng quát và đưa việc đánh giá đúng/sai vào ngôn ngữ của mình. Hàm bool() trong Python sẽ đánh giá đúng/sai cho tất cả các đối tượng dữ liệu của mình.

Xem các ví dụ sau để biết rõ hơn hàm bool() đánh giá các đối tượng khác nhau.

```
>>> bool(0)          # số 0 có giá trị bool là False
False
>>> bool(4)          # số 4 có giá trị bool là True
True
>>> bool([])         # list rỗng có giá trị False
False
>>> bool((1, 2, 3))  # dãy (1, 2, 3) có giá trị True
True
```

Như vậy hàm `bool()` trong Python rất đặc biệt, có thể đánh giá đúng/sai cho tất cả các đối tượng từ đơn giản như số, xâu đến phức tạp như list, tuple hay dữ liệu tập hợp, từ điển mà chúng ta sẽ học trong các bài học sau.

Không những vậy hàm `bool()` còn đánh giá được cả các biểu thức tính toán phức tạp, kể cả các hàm số (không phân biệt có trả lại giá trị hay không).

Tính năng này của Python giúp chúng ta rất nhiều trong quá trình lập trình, thiết kế thuật toán, làm cho chương trình có thể rút ngắn, đơn giản hơn rất nhiều, mặc dù điều này có khác nhiều so với một số ngôn ngữ lập trình bậc cao khác.

Bảng sau cho chúng ta qui định tính toán hàm `bool()` với các đối tượng dữ liệu bất kỳ.

Giá trị	Đối tượng nhận giá trị logic tương ứng
False	Các đối tượng số = 0, biểu thức nhận giá trị 0, None, list, tuple rỗng, xâu ký tự rỗng, hàm số void hoặc hàm số trả lại giá trị 0.
True	Các đối tượng số khác 0, biểu thức nhận giá trị khác 0, list, tuple không rỗng, xâu ký tự không rỗng, hàm số trả lại giá trị khác 0.

Một số ví dụ minh họa đơn giản của hàm `bool()`.

```
>>> A = []
>>> bool(A)
False
>>> A.append(5)
>>> bool(A)
True
>>> bool(12*3 - 35)
True
>>> bool(3**3 - 12 - 15)
False
>>>
```

Ví dụ sau minh họa hàm `bool()` tính toán trên các hàm số dựa vào giá trị trả lại của hàm,

```
1. >>> a = 10
2. >>> b = 11
3. >>> def f(a,b):
4.     return a + 1 - b
5.
6. >>> f(a,b)
7. 0
8. >>> bool(f(a,b))
9. False
10. >>> a = 2
11. >>> bool(f(a,b))
12. True
```

Chúng ta đã thấy khái niệm dữ liệu logic trong Python đã được mở rộng rất nhiều và có ứng dụng rất lớn. Nếu coi mọi loại, kiểu dữ liệu đều là một dạng của logic thì có thể thực hiện các phép toán logic (như and, or, not) trên một lớp lớn các đối tượng dữ liệu khác nhau. Ví dụ:

```
>>> bool(12 and [1, 2, 3])
True
>>> bool(33 and "")
False
>>>
```

Để kết thúc phần giới thiệu mở đầu cho kiểu dữ liệu logic này chúng ta cùng thực hiện một số bài toán, ứng dụng ban đầu.

1. Cho trước một dãy (list) A với các phần tử đa dạng khác nhau. Cần loại bớt khỏi A các phần tử “vô nghĩa”, ví dụ các số 0, list hay tuple rỗng, các đối tượng None, xấu rỗng.

Ví dụ A được định nghĩa như sau:

```
A = [1, 0, None, "One", "", (), [1, 2, 3], [], "Two", "Three",
20, 30, [0, 0], 0, 0]
```

Khi đó đoạn chương trình ngắn sau sẽ lọc bớt các phần tử “vô nghĩa” trong A, kết quả đưa vào danh sách B.

```
>>> A = [1, 0, None, "One", "", (), [1, 2, 3], [], "Two", "Three",
20, 30, [0, 0], 0, 0]
>>> B = []
>>> for item in A:
    if item:
        B.append(item)
>>> B
[1, 'One', [1, 2, 3], 'Two', 'Three', 20, 30, [0, 0]]
>>>
```

2. Cho trước một dãy (list) A gồm các số khác nhau. Cần tính trung bình cộng của các phần tử khác 0 của A (tức là các phần tử = 0 sẽ bị loại ra không tính vào trung bình cộng).

```
>>> A = [1, 5.3, 9+0j, 0, 0, 12*6, 33 - 12*5j, 0, 1, 2, 3]
>>> def avg(A): # thiết lập hàm số tính trung bình cộng
    count = 0
    S = 0
    for x in A:
        if x: # kiểm tra xem x có khác 0?
            count = count + 1
            S = S + x
    return S/count
>>> avg(A)
8.2875
>>>
```

3. Cho trước một dãy (list) A gồm các số, trong các số này có thể có nhiều số lặp lại giống nhau. Yêu cầu loại bớt các cặp số liền nhau giống nhau trong dãy trên.

Lời giải có thể như chương trình sau, chú ý đến dòng lệnh 4.

```

1. A = [0,0,2,1,3,3,3, 8,8, 0,0,-1,-1,10,10,10,10]
2. B = [A[0]]
3. for i in range(1,len(A)):
4.     if A[i] - A[i-1]:
5.         B.append(A[i])
6. print(B)

```

Với mảng A như trên thì kết quả sẽ thu được dãy B như sau:

```
[0, 2, 1, 3, 8, 0, -1, 10]
```

2. Biểu thức logic và các toán tử so sánh

Nhắc lại ở đây bảng các toán tử so sánh cơ bản của kiểu dữ liệu logic mà chúng ta đã biết từ Chương 3. Các phép so sánh này sẽ tạo ra các đối tượng dữ liệu kiểu bool và dùng rất nhiều trong các bài toán lập trình, trong các lệnh điều khiển có điều kiện như **if** hay **while**.

2.1. Phép so sánh trong Python

Phép so sánh	Ý nghĩa
x == y	x có bằng y hay không?
x != y	x có khác y hay không?
x > y	x có lớn hơn y hay không?
x >= y	x có lớn hơn hoặc bằng y hay không?
x < y	x có nhỏ hơn y hay không?
x <= y	x có nhỏ hơn hoặc bằng y hay không?

Bên cạnh bảng trên, Python còn hỗ trợ cách so sánh bắc cầu, gộp rất đa dạng như sau.

```

>>> a = 1
>>> 0 < a <= 1 # a có lớn hơn 0 và nhỏ hơn hoặc bằng 1 không?
True
>>> 1 < a <= 2 # a có lớn hơn 1 và nhỏ hơn hoặc bằng 2 không?
False
>>>

```

Tổng quát phép so sánh có thể gộp lại và bắc cầu theo cú pháp sau:

```
<obj1> <op1> <obj2> <op2> ... <objN> <opN> <obj{N+1}>
```

Quan sát các ví dụ sau để hiểu rõ hơn về cách viết các phép so sánh gộp như thế này.

```

>>> a = 1
>>> b = 2
>>> c = 3
>>> 1 <= a < 2 >= b
True
>>> 3 >= c > 2 > b >= 1 >= a
False
>>> 3 >= c > 2 >= b > 1 >= a > 0
True
>>>

```

2.2. Các toán tử logic trong Python

Có 3 phép toán chính với kiểu dữ liệu bool là **and** (và), **or** (hoặc) và **not** (phủ định). Bảng giá trị của các phép toán này được định nghĩa như sau.

Phép toán và **and**

x	and	y	x and y
True	and	True	True
False	and	False	False
True	and	False	False
False	and	True	False

Phép toán hoặc **or**

x	or	y	x or y
True	or	True	True
False	or	False	False
True	or	False	True
False	or	True	True

Phép toán phủ định **not**

x	not x
True	False
False	True

Kết hợp các phép toán logic trên với các phép so sánh gộp trong mục trên, cùng với tính chất có thể coi mọi đối tượng, biểu thức là logic sẽ làm cho Python trở nên rất linh hoạt, đa dạng trong việc giải các bài toán. Chúng ta cùng quan sát thông qua một số ví dụ cụ thể sau đây.

1. Viết hàm số kiểm tra xem giá trị năm có phải là năm nhuận hay không. Năm nhuận là năm thỏa mãn tính chất sau: Năm đó chia hết cho 400 hoặc chia hết cho 4 nhưng không chia hết cho 100. Giá trị trả lại của hàm là True hoặc False.

Hàm kiểm tra năm nhuận có thể viết như sau:

```
1. def nhuan(n):
2.     if n % 400 == 0 or (n % 4 == 0 and n % 100 != 0):
3.         return True
4.     else:
5.         return False
```

Kết quả của hàm cho một số giá trị năm:

```
>>> nhuan(2000)
True
>>> nhuan(2100)
False
>>>
```

2. Điểm trung bình các môn của học sinh được quy định đánh giá học lực như sau:

- Nếu điểm < 3 → học lực kém.
- Từ 3 đến < 5 → học lực yếu.
- Từ 5 đến < 7 → học lực trung bình.
- Từ 7 đến < 9 → học lực khá.

- Từ 9 trở lên → học lực giỏi.

Cho biết điểm học sinh, yêu cầu ghi ra học lực học sinh đó.

Lời giải bài toán trên có thể như sau.

Hàm kiểm tra học lực theo điểm.

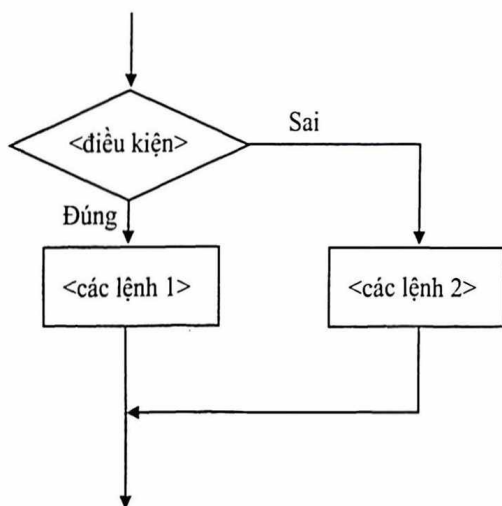
```
def hocluc(d):  
    if d < 3:  
        return "học lực kém"  
    if 3 <= d < 5:  
        return "học lực yếu"  
    if 5 <= d < 7:  
        return "học lực trung bình"  
    if 7 <= d < 9:  
        return "học lực khá"  
    if d >= 9:  
        return "học lực giỏi"
```

3. Lệnh if

Chúng ta đã được học và biết về lệnh điều khiển rẽ nhánh `if ... else ...` trong Python. Cú pháp của lệnh như sau.

```
if <điều kiện>:  
    <các lệnh 1>  
else:  
    <các lệnh 2>
```

Khi thực hiện lệnh trên, chương trình sẽ kiểm tra <điều kiện>, nếu đúng sẽ thực hiện nhóm <các lệnh 1>, còn nếu sai thì thực hiện <các lệnh 2>. Mô hình lệnh `if` như trên gọi là cấu trúc rẽ nhánh (2 nhánh) biểu diễn trên sơ đồ như sau.



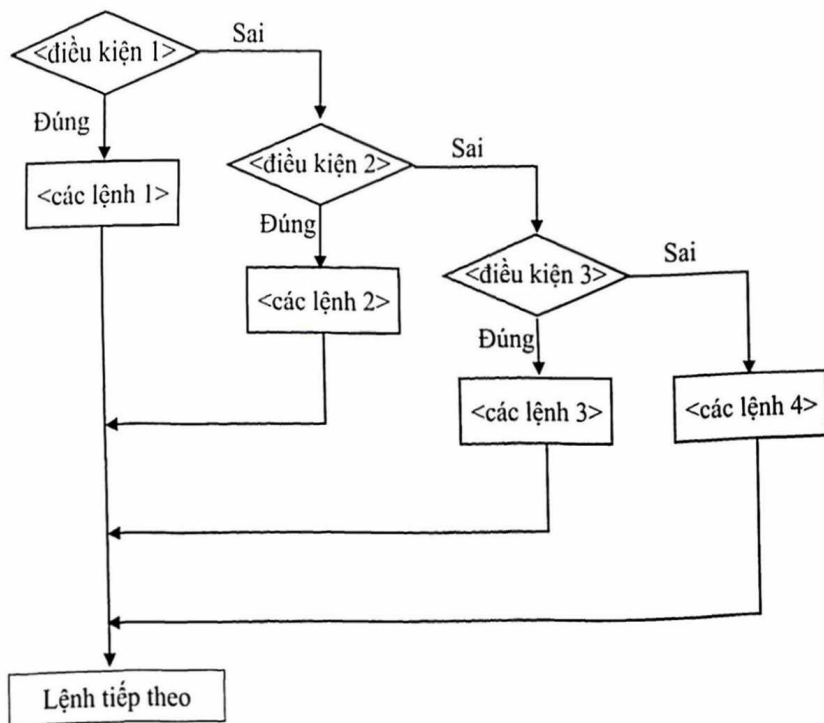
Các lệnh `if` có thể lồng nhau khi cần có nhiều rẽ nhánh (số lượng nhánh > 2). Quay trở lại bài toán tính học lực của học sinh theo điểm trung bình, bài toán này có 5 nhánh. Chúng

ta có thể dùng các lệnh **if** lồng nhau để thể hiện bài toán này với chỉ một lệnh **if** duy nhất. Xem chương trình sau trong đó dùng một lệnh **if** duy nhất nhưng bên trong lệnh này có 5 lớp lồng nhau. Các rẽ nhánh tiếp theo được thể hiện bằng lệnh **if** và nằm trong vùng từ khóa **else** của lệnh.

```
def hocluc(d):  
    if d < 3:  
        return "học lực kém"  
    else:  
        if 3 <= d < 5:  
            return "học lực yếu"  
        else:  
            if 5 <= d < 7:  
                return "học lực trung bình"  
            else:  
                if 7 <= d < 9:  
                    return "học lực khá"  
                else:  
                    if d >= 9:  
                        return "học lực giỏi"
```

Mô hình rẽ nhiều nhánh như trên gặp rất nhiều trên thực tế.

Sơ đồ sau mô tả mô hình lệnh rẽ 4 nhánh nhưng được thể hiện bởi lồng ghép các sơ đồ 2 nhánh. Tương tự như mô hình các lệnh **if** lồng ghép nhau ở trên.



Python cho phép thực hiện cấu trúc lệnh **if** với nhiều rẽ nhánh bằng từ khóa **elif**. Từ khóa này chính là cấu trúc **else if** nhưng được viết chỉ bằng một lệnh. Ví dụ với bài toán tính học lực có thể viết như sau.

Chương trình sau được viết lại theo mô hình lệnh **if** đầy đủ có cấu trúc rẽ nhiều nhánh với bộ 3 **if - elif - else**, trong đó từ khóa **elif** có thể lặp lại nhiều lần.

```
1. def hocluc(d):
2.     if d < 3:
3.         return "học lực kém"
4.     elif 3 <= d < 5:
5.         return "học lực yếu"
6.     elif 5 <= d < 7:
7.         return "học lực trung bình"
8.     elif 7 <= d < 9:
9.         return "học lực khá"
10.    else:
11.        return "học lực giỏi"
```

Cú pháp tổng quát của lệnh **if** trong Python với cấu trúc rẽ nhiều nhánh như sau.

```
if <điều kiện 1>:
    <các lệnh 1>
elif <điều kiện 2>:
    <các lệnh 2>
elif ...
...
elif ...
...
else:
    <các lệnh N>
```

Trong cấu trúc của lệnh **if** tổng quát, nếu có N rẽ nhánh thì cần N-2 câu lệnh **elif**.

4. Lệnh lặp **while**

Chúng ta đã học lệnh lặp **while** trong Chương 3. Đây là dạng lệnh lặp có điều kiện, lặp mà không biết trước số vòng lặp. Số vòng lặp phụ thuộc vào một điều kiện cho trước. Điều kiện này sẽ kiểm soát sự dừng lại của vòng lặp.

Cú pháp của lệnh lặp **while** như sau.

```
while <điều kiện>:
    <các lệnh lặp>
```

Chú ý: Lệnh lặp **while** có thể tạo ra vòng lặp mãi mãi, tức là lặp không bao giờ dừng. Ví dụ đoạn chương trình sau:

```
while True:
    print("Welcome!")
```

Nếu rơi vào trường hợp chương trình bị lặp vô hạn như trên thì có thể dùng tổ hợp phím Ctrl+C để ngắt chương trình.

Bây giờ chúng ta sẽ mô tả khuôn dạng tổng quát nhất của lệnh **while** trong Python. Cú pháp lệnh này như sau.

```
while <điều kiện>:  
    <các lệnh lặp>
```

```
else:  
    <các lệnh sau cùng>
```

Ý nghĩa câu lệnh **while** tổng quát trên như sau: Chương trình sẽ luôn kiểm tra <điều kiện>, nếu đúng thì thực hiện nhóm <các lệnh lặp> và quay lại kiểm tra <điều kiện>, nếu sai thì thực hiện các lệnh trong nhóm <các lệnh sau cùng> và kết thúc lệnh.

Xét ví dụ đơn giản sau.

```
n = 10  
while n > 0:  
    print(n)  
    n = n - 1  
else:  
    print("kết thúc")
```

Kết quả chạy chương trình trên như sau:

```
10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
kết thúc  
>>>
```

5. break và continue

Các từ khóa **break** hay **continue** có thể đặt giữa các lệnh lặp như một lệnh có tác dụng đặc biệt. Các từ khóa này có thể có trong các lệnh lặp **for** và **while**. Ý nghĩa các lệnh đặc biệt này như sau:

- Nếu gặp lệnh **break**, chương trình sẽ lập tức kết thúc lệnh lặp hiện thời.
- Nếu gặp lệnh **continue**, chương trình sẽ kết thúc vòng lặp tại vị trí đang đứng và quay trở lại đầu vòng lặp tiếp theo.

1. Chương trình yêu cầu nhập liên tục các số, sau đó in các số đó ra màn hình, chương trình chỉ dừng lại khi người dùng nhập số 0.

VonglapWhile.py

```
1. while True:  
2.     x = eval(input("Nhập số bất kỳ:"))  
3.     if x:  
4.         print(x)  
5.     else:  
6.         break
```

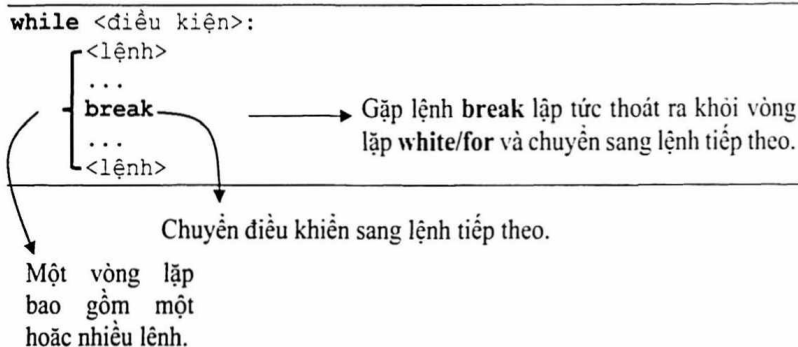
2. Ví dụ sau sẽ yêu cầu nhập liên tục các tên học sinh trong lớp, yêu cầu phải nhập với chữ cái đầu tiên viết Hoa. Nếu nhập sai thì thông báo để người dùng nhập tiếp. Chương trình sẽ dừng lại khi người dùng không nhập gì cả mà chỉ bấm Enter.

Nhap_DSHS.py

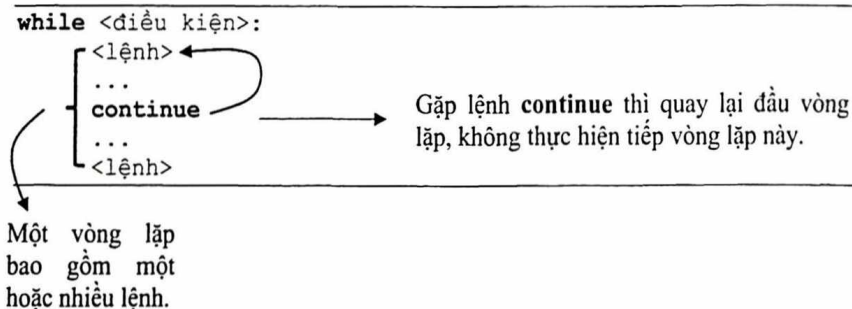
```
1. count = 0
2. while True:
3.     name = input("Hãy nhập một tên học sinh: ")
4.     if not name:
5.         break
6.     if name[0].islower():
7.         print("Nhập sai rồi, hãy nhập lại chữ cái đầu cần viết
   hoa.")
8.         continue
9.     else:
10.        print("Tên học sinh đã nhập:", name)
11.        count = count + 1
12. print("Chúc mừng bạn đã nhập", count, "học sinh.")
```

Sơ đồ mô tả 2 từ khóa trên như sau.

Từ khóa **break**.



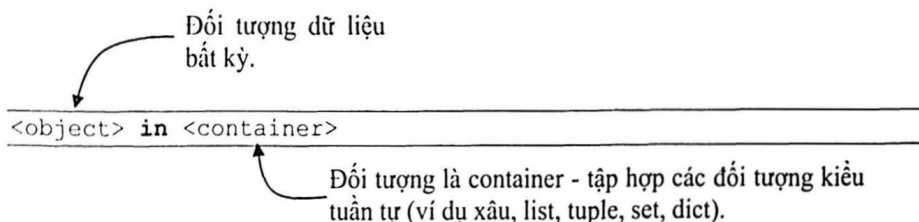
Từ khóa **continue**.



Chú ý: Các lệnh **break** và **continue** cũng áp dụng hoàn toàn tương tự với lệnh lặp **for**.

6. Toán tử in

Toán tử **in** cho phép thiết lập thêm một quan hệ logic nữa giữa các đối tượng trong Python. Toán tử này cho chúng ta biết được một đối tượng nào đó có nằm trong một đối tượng khác hay không. Đối tượng khác này có dạng kiểu container (cái thùng) có thể chứa các dữ liệu khác, ví dụ kiểu tuần tự, **string**, **list**, **tuple**, **set**, **dict**⁶.



Ví dụ đối với list.

```
>>> A = [1, 2, 3, 4]
>>> b = 0 in A
>>> b
False
>>>
```

Không chỉ có **list** hay **tuple**, toán tử **in** còn có tác dụng với các kiểu dữ liệu tuần tự khác như **xâu ký tự**, thậm chí cả **range**.

```
>>> 0 in range(10)
True
>>> 12 in range(5)
False
>>> "Hà" in "Hà Nội"
True
>>>
```

7. Nguyên tắc chuyển mạch nhanh của các phép toán logic

Chúng ta cùng tìm hiểu sâu hơn cơ chế làm việc của các phép toán logic **and** (và) và **or** (hoặc) trong Python. Các phép toán này được gọi là phép toán chuyển mạch nhanh (short-circuit operator). Một trong các đặc thù đặc biệt nhất của 2 phép toán này là cơ chế (nguyên tắc) tính nhanh của chúng. Python không cần tính tất cả các tham số (toán hạng) của các phép toán mà vẫn có thể tính nhanh chính xác kết quả của phép toán đó. Cơ chế tính nhanh này gọi là nguyên tắc chuyển mạch nhanh của các phép toán **and**, **or** trong Python. Người lập trình Python nâng cao rất cần hiểu cơ chế này của Python để thực hiện các công việc gỡ rối phức tạp khi viết các chương trình lớn.

Vì sao rất cần biết sâu về cơ chế chuyển mạch nhanh này của Python?

Vấn đề là đặc tính có thể đưa mọi đối tượng, biểu thức, hàm số về đánh giá logic bằng hàm `bool()` của Python. Chính tính năng này làm cho đôi khi việc tìm lỗi chương trình của Python trở nên rất khó khăn. Do vậy tìm hiểu sâu về cơ chế chuyển mạch nhanh của

⁶ Kiểu dữ liệu `set` và `dict` sẽ được học trong các bài học tiếp sau của sách này.

Python là rất cần thiết và hữu ích cho những ai muốn tìm hiểu lập trình Python chuyên sâu và nâng cao.

Chúng ta sẽ lần lượt tìm hiểu các nguyên tắc này đối với từng phép toán `and`, `or`.

Trước tiên chúng ta tìm hiểu phép toán `and`.

x	and	y	x and y
True	and	True	True
False	and	False	False
True	and	False	False
False	and	True	False

Với phép toán `and`, chúng ta thấy chỉ cần 1 toán hạng là `False` thì lập tức kết quả sẽ là `False` mà không phụ thuộc vào toán hạng thứ 2. Do vậy nếu biết 1 toán hạng là `False` thì không cần tính toán hạng thứ 2 nữa.

Xem kỹ 2 đoạn lệnh sau.

```
>>> 2 and input("Nhập số:")
Nhập số:3
```

→ Bắt buộc phải tính toán hạng 2

```
>>> 0 and input("Nhập số:")
0
```

→ Không cần tính toán hạng 2

Nhận xét:

- Ở lệnh đầu, Python biết tham số đầu là 2, biến số này có giá trị `True`, do đó Python bắt buộc phải thực hiện toán hạng thứ 2 là `input("Nhập số:")` để kiểm tra xem nó là `True` hay `False`. Do vậy cần thực hiện lệnh `input("Nhập số:")` và kết quả ở 2 dòng phía dưới.

- Cùng xem đoạn lệnh thứ 2, tham số đầu là 0, tức là `False`, do tính chất của phép logic `and`, giá trị của biểu thức này lập tức là `False` mà không cần thực hiện toán hạng thứ 2 là `input("Nhập số:")`.

Xét tiếp ví dụ sau, gán `x` cho một biểu thức logic.

```
1. >>> x = False and print(1)
2. >>> x
3. False
4. >>> x = True and print(1)
5. 1
6. >>> x
7. >>> print(x)
8. None
```

Nhận xét:

- Với dòng lệnh 1, khi gán `x = False and print(1)`, vì toán hạng đầu tiên là `False` nên Python sẽ không cần tính `print(1)` nữa và gán ngay `x` bằng `False`. Kết quả thể hiện ở các dòng 2, 3.

- Với dòng lệnh 4, `x = True and print(1)`, vì toán hạng đầu tiên là `True` nên Python bắt buộc phải tính toán hạng thứ 2 là `print(1)` và gán `x = print(1)`. Kết quả dòng 6, `x` không trả lại giá trị nào vì `x` có kiểu `NoneType`. Do vậy khi thực hiện lệnh tại dòng 7, `print(x)` sẽ trả lại giá trị `None`.

Bây giờ chúng ta tìm hiểu phép toán **or**.

x	or	y	x or y
True	or	True	True
False	or	False	False
True	or	False	True
False	or	True	True

Với phép toán **or**, chúng ta thấy chỉ cần 1 toán hạng là True thì lập tức kết quả sẽ là True mà không phụ thuộc vào toán hạng thứ 2. Do vậy nếu biết 1 toán hạng là True thì không cần tính toán hạng thứ 2 nữa.

Quan sát các ví dụ sau.

```
1. >>> print("Hi") or print(3)
2. Hi
3. 3
4. >>> [0] or print(1)
5. [0]
6. >>>
```

- Với lệnh đầu tiên (dòng 1), toán hạng đầu tiên `print("Hi")` nhận giá trị logic False, do đó cần phải tính thêm cả toán hạng thứ 2. Do vậy kết quả là Python thực hiện cả 2 toán hạng, thể hiện tại dòng 2, 3.

- Với dòng lệnh 4, vì toán hạng thứ nhất `[0]` nhận giá trị True, do đó không cần đến toán hạng thứ 2 nữa, chương trình chỉ hiện kết quả `[0]` là toán hạng 1 tại dòng 5.

Chúng ta cùng xem lệnh gán x với phép toán **or**.

```
1. >>> x = [0] or [1, 1]
2. >>> x
3. [0]
4. >>> x = [] or [1, 2]
5. >>> x
6. [1, 2]
7. >>>
```

Nhận xét:

- Với lệnh đầu tiên (dòng 1), x được gán với biểu thức `[0] or [1, 1]`. Vì toán hạng đầu tiên là `[0]` là True nên không cần tính phần tử thứ 2 nữa và Python gán luôn `[0]` cho x. Kết quả thể hiện ở các dòng 2, 3.

- Với lệnh thứ 2 (dòng 4), x được gán với biểu thức `[] or [1, 2]`. Vì `[]` là False nên bắt buộc tính thêm toán hạng thứ 2, do đó Python sẽ gán toán hạng 2 là `[1, 2]` cho x. Kết quả thể hiện ở các dòng 5, 6.

Xét thêm một ví dụ nữa để hiểu nguyên tắc chuyển mạch nhanh này.

```
1. >>> x = False or input("Nhập số:")
2. Nhập số:3
3. >>> x
4. '3'
```


Với lệnh trên, x được gán với một biểu thức logic là False or input("Nhập số:"). Vì toán hạng đầu tiên là False nên bắt buộc phải tính đến toán tử thứ 2, do đó Python sẽ gán toán hạng này cho x, tức là gán x = input("Nhập số:"). Kết quả là lệnh này được thực hiện tại dòng số 2, người dùng nhập số 3, sau đó x sẽ nhận giá trị 3, kết quả thể hiện ở dòng 4.

Các ví dụ sau còn hay hơn nữa. Các ví dụ này đều xuất phát chính là ví dụ trên nhưng với các cách tương tác khác nhau của người dùng, giá trị của biểu thức được gán sẽ nhận các giá trị khác nhau.

```
1. >>> x = False or input("Nhập số:")
2. Nhập số:
3. >>> bool(x)
4. False
5. >> x = False or input("Nhập số:")
6. Nhập số:3
7. >>> bool(x)
8. True
```

- Với dòng lệnh đầu tiên, chúng ta đã biết, Python sẽ gán x cho lệnh input("Nhập số:"), nếu người dùng nhập xâu rỗng (chỉ nhấn Enter, không gõ gì) thì kết quả nhận được là xâu rỗng, kết quả này thể hiện tại dòng 3. Do đó giá trị bool của x là False.

- Với dòng lệnh tiếp theo (dòng 5), vẫn với lệnh đó, người dùng nhập số 3, khi đó x sẽ nhận giá trị là ký tự "3", và giá trị bool của x là True.

Do vậy giá trị của biểu thức:

```
False or input("Nhập số:")
```

sẽ phụ thuộc vào tương tác với người dùng chứ không phụ thuộc vào bản thân biểu thức logic trên. Đây có lẽ là đặc thù rất đặc biệt của Python, xuất phát từ nguyên tắc chuyển mạch nhanh mà chúng ta vừa biết.



Câu hỏi - bài tập

1. Giá trị x sau khi thực hiện lệnh sau sẽ là gì?

```
x = 4 ** 2 - 4 * 4 == 0
```

- a) True
- b) False
- c) Không là False và cũng không là True
- d) Chương trình báo lỗi.

2. Cho đoạn chương trình:

```
>>> a = 5
>>> b = 7
>>> X = a < b or a < 0 and b < 0
```

Giá trị của X là:

- a) True
- b) False
- c) Chương trình báo lỗi.

3. Cho đoạn chương trình:

```
>>> a = 5
>>> b = 7
>>> X = (a < b or a < 0) and (b < 0)
```

Giá trị của X là:

- a) True
- b) False
- c) Chương trình báo lỗi.

4. Cho đoạn chương trình sau:

```
1. A = [1, 15, -9, 20, -99, 100, 120, 7, 10]
2. B = []
3. for i in A:
4.     if i > 1 and i < 15:
5.         B.append(i)
6. print(B)
```

Dãy B có bao nhiêu phần tử?

- a) 1
- b) 2
- c) 5
- d) 9

5. Viết thủ tục khai triển một số tự nhiên thành tích các thừa số nguyên tố.

- Tham số đầu vào: Số tự nhiên n.

- Đầu ra: In trên màn hình dãy các ước số nguyên tố của n trên một dòng, cách nhau bởi dấu cách.

Ví dụ:

Nếu n = 10 thì in ra 2 5.

Nếu n = 8 thì in ra 2 2 2 2.

6. Viết thủ tục khai triển một số tự nhiên thành tích các thừa số nguyên tố với số mũ đầy đủ.

- Tham số đầu vào: số tự nhiên n.

- Đầu ra: in trên màn hình dãy các ước số nguyên tố của n trên một dòng, cách nhau bởi dấu cách. Mỗi số nguyên tố ghi một lần và sau đó là số lần thể hiện thừa số này, ghi trong dấu ngoặc.

Ví dụ:

Nếu n = 10 thì in ra 2 5.

Nếu n = 8 thì in ra 2 (4).

Nếu n = 100 thì in ra 2 (2) 5 (2).

Chú ý: Có thể thay đổi cách thể hiện, dùng dấu ^ để chỉ số mũ, ví dụ 2² 5².

7. Viết hàm số **NextPrime(n)** có ý nghĩa như sau:

- Tham số là số tự nhiên n .
- Hàm trả lại số nguyên tố nhỏ nhất lớn hơn n .

8. Đoạn chương trình sau thực hiện công việc gì?

```
>>> def f(A):  
    for i in range(len(A) - 1):  
        if A[i] > A[i+1]:  
            return False  
    return True  
>>> A = [2, 1, 3]  
>>> f(A)
```

Giá trị của $f(A)$ trong chương trình trên là gì?

- a) True
- b) False
- c) Chương trình báo lỗi.

9. Cho trước dãy (list) A gồm các số bất kỳ.

Viết chương trình tính chỉ số và giá trị phần tử lớn nhất của dãy A .

10. Cho trước dãy (list) A gồm các số.

Viết chương trình tính chỉ số và giá trị phần tử nhỏ nhất của dãy A .

11. Nhập số n từ bàn phím và tính tổng $S = 1/1^2 + 1/2^2 + \dots + 1/n^2$.

12. Nhập số n từ bàn phím và tính tổng $S = 1/(1.2) + 1/(2.3) + \dots + 1/(n.(n+1))$.

13. Tính tổng các số của dãy sau:

$$1 + 5 + 9 + 13 + \dots + 205.$$

14. Nhập số n từ bàn phím, in ra các số nguyên tố nhỏ hơn n .

15. Nhập số n từ bàn phím, in ra khai triển của số n thành tích các thừa số nguyên tố. Ví dụ $20 = 2 \times 2 \times 5$.

16. Nhập số n , in ra n số hạng đầu tiên của dãy số Fibonacci.

17. Nhập 2 số tự nhiên n, m , tính ước số chung lớn nhất (USCNN) và bội số chung nhỏ nhất (BSCNN) của n, m .

18. Nhập 3 số tự nhiên n, m, p tính ước số chung lớn nhất của n, m, p .

19. Nhập 2 số tự nhiên n, m , tính và đếm số các ước chung của n, m .

20. Nhập 3 số tự nhiên n, m, p tính bội số chung nhỏ nhất của n, m, p .

21. Nhập số n và dãy số $a_1, a_2, \dots, a_n$ từ bàn phím. Tính:

a) Tổng các số của dãy trên.

b) Tìm phần tử (chỉ số) tương ứng với số lớn nhất và nhỏ nhất của dãy trên.

22. Nhập số n và dãy số $a_1, a_2, \dots, a_n$ từ bàn phím. Tìm một dãy con liên tục cực đại đơn điệu tăng của dãy số trên.

23. Tương tự bài 22. Tìm một dãy con liên tục cực đại các số > 0 của dãy trên.

24. Nhập số n từ bàn phím, tìm và in ra tất cả các ước số nguyên tố của số n .
25. Nhập 3 số m, n, p từ bàn phím, tìm và in ra tất cả các ước số chung của 3 số trên.
26. Số hoàn hảo là số = tổng các ước số thực sự của số đó. Tìm tất cả các số hoàn hảo nhỏ hơn 10000.
27. Nhập số n từ bàn phím, tìm:
- Số nguyên tố nhỏ nhất $> n$.
 - Số nguyên tố lớn nhất $< n$.
 - Số nguyên tố nhỏ nhất $> n/2$.
 - Số nguyên tố lớn nhất $< n/2$.
28. Nhập từ bàn phím số n , hãy viết n trong hệ nhị phân.
29. Nhập từ bàn phím số n , hãy viết n trong hệ hex (hệ đếm 16).
30. Nhập số n và dãy số $a_1, a_2, \dots, a_n$ từ bàn phím. Tìm ra các phần tử cực trị địa phương của dãy này (phần tử cực trị địa phương nếu nó cùng lớn hơn hoặc nhỏ hơn 2 số bên cạnh).
31. Viết hàm số **PrevPrime(n)** có ý nghĩa như sau:
- Tham số là số tự nhiên n .
 - Hàm trả lại số nguyên tố lớn nhất nhỏ hơn hoặc bằng n . Nếu $n = 1$ thì trả lại False.
32. Viết chương trình thực hiện công việc sau:
- Yêu cầu người dùng liên tục nhập các số nguyên từ bàn phím. Quá trình nhập chỉ kết thúc khi người dùng nhấn phím ENTER khi chưa gõ bất cứ phím nào.
 - Các số đã nhập sẽ được lưu vào một dãy, sau đó thể hiện trên màn hình.
33. Viết chương trình thực hiện công việc sau:
- Yêu cầu người dùng liên tục nhập các số nguyên khác nhau đôi một từ bàn phím. Quá trình nhập chỉ kết thúc khi người dùng nhấn phím ENTER khi chưa gõ bất cứ phím nào.
 - Nếu người dùng nhập một số đã được nhập từ trước thì thông báo ngay "Số này bạn đã nhập rồi" và yêu cầu nhập tiếp.
 - Khi kết thúc chương trình in các số đã nhập (khác nhau) ra màn hình.
34. Khi viết lệnh này kết quả sẽ là gì?
-
- ```
x = 2 < 1 or print("hi")
```
- 
35. Khi viết lệnh này kết quả sẽ là gì?
- 
- ```
x = print() or print("hi")
```
-
36. Khi viết lệnh này kết quả sẽ là gì?
-
- ```
x = print("ha") or print("hi")
```
- 
37. Khẳng định sau đúng hay sai? Nếu lý do và giải thích.
- Một mệnh đề logic (mệnh đề có các toán tử **and, or, not**) có thể không xác định được ngay giá trị vì còn phụ thuộc vào tương tác với người sử dụng.

## Chương 12. Đệ quy

### Mục tiêu:

- Khái niệm đệ quy và hàm gọi đệ quy.
- Cách thiết lập hàm đệ quy trong Python và một vài ứng dụng.



### Nội dung chính

Đệ quy, tên tiếng Anh là **recursion** - sự tự lặp lại, là một kỹ thuật lập trình rất nổi tiếng với ý tưởng gốc là chia một bài toán lớn thành các bài toán nhỏ hơn để thực hiện. Kỹ thuật gốc của khái niệm này xuất phát từ tên gọi của nó là sự tự lặp lại. Phát biểu bằng ngôn ngữ lập trình là sự gọi lại chính mình. Một trong những khái niệm nổi tiếng nhất của Toán học là dãy số Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

được định nghĩa như sau:

$$F_1 = 1$$

$$F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

Công thức  $F(n) = F(n-1) + F(n-2)$  dùng để tính số hạng Fibonacci tổng quát chính là cách định nghĩa đệ quy của dãy này. Bài toán tính dãy Fibonacci là ví dụ điển hình của khái niệm đệ quy.

### 1. Khái niệm hàm đệ quy

Như vậy hàm số được gọi là đệ quy nếu trong định nghĩa của hàm có lời gọi chính nó. Chúng ta cùng viết ví dụ đầu tiên, đơn giản về một hàm số đệ quy.

Hàm đệ quy đơn giản `rc1()`.

---

```
>>> def rc1():
 rc1()
```

---

Khi chạy thử, sẽ có thông báo lỗi của Python như sau.

---

```
>>> rc1()
Traceback (most recent call last):
 File "<pyshell#45>", line 1, in <module>
 rc1()
 File "<pyshell#44>", line 2, in rc
 rc1()
 File "<pyshell#44>", line 2, in rc
 rc1()
 File "<pyshell#44>", line 2, in rc
 rc1()
[Previous line repeated 990 more times]
RecursionError: maximum recursion depth exceeded
```

---



Lỗi ở đây là dòng lệnh thứ 2, bên trong hàm `rc1()` bị gọi liên tục quá 990 lần, vượt qua ngưỡng qui định số lần lặp lại của một lệnh trong Python.

Như vậy khi viết hàm đệ quy phải rất cẩn thận chỗ này. Xem lại chương trình trên lý do của việc lời gọi lại chính mình của hàm bị lặp liên tục là do hàm không có tham số.

Chúng ta cùng xét ví dụ tiếp theo.

---

```
>>> def rc2(n):
 print("Next:", n)
 rc2(n+1)
 print("Stop")
```

---

Hàm `rc2()` có một tham số `n`, trong lời gọi bên trong tham số này đã thay đổi.

Khi chạy Python hiện thông báo như sau.

---

```
>>> rc2(1)
Next: 1
Next: 2
Next: 3
Next: 4
.
<xóa các dòng từ 5 đến 975 tại đây>
.
Next: 976
Next: 977
Next: 978
Next: 979
Traceback (most recent call last):
 File "<pyshell#65>", line 1, in <module>
 rc2(1)
 File "<pyshell#64>", line 3, in rc2
 rc2(n+1)
 File "<pyshell#64>", line 3, in rc2
 rc2(n+1)
 File "<pyshell#64>", line 3, in rc2
 rc2(n+1)
[Previous line repeated 975 more times]
 File "<pyshell#64>", line 2, in rc2
 print("Next:", n)
RecursionError: maximum recursion depth exceeded while pickling
an object
>>>
```

---

Vẫn là lỗi lặp lại một lệnh quá ngưỡng cho phép.

Chú ý: Hàm `getrecursionlimit()` trong module `sys` sẽ cho chúng ta biết ngưỡng giới hạn cho phép của một lệnh được phép lặp lại liên tục là bao nhiêu. Giới hạn này mặc định là 1000 lần.

---

```
>>> import sys
>>> sys.getrecursionlimit()
1000
>>>
```

---

## 2. Thiết lập đệ quy đúng

Từ 2 ví dụ trên chúng ta thấy nếu không thiết kế cẩn thận, các hàm đệ quy sẽ vướng lỗi lặp mãi mãi. Cần thiết lập đệ quy đúng. Sau đây là 2 nguyên tắc cơ bản khi viết các hàm đệ quy cần nhớ.

1. Tham số của hàm đệ quy cần thay đổi trong lời gọi lặp bên trong của hàm.
2. Cần có điểm dừng cho lặp đệ quy, hay nói cách khác bên trong hàm đệ quy cần có chỗ cho việc điều khiển dừng, kết thúc lời gọi đệ quy.

Các ví dụ trong phần trên của bài học đều không thỏa mãn một hoặc cả 2 nguyên tắc trên. Chúng ta sẽ phân tích và thiết lập một số hàm đệ quy đúng trong phần này.

### 2.1. Hàm tính dãy Fibonacci

Công thức tính dãy số Fibonacci tổng quát chính là lời gọi đệ quy của hàm số này:

$$F_n = F_{n-1} + F_{n-2}$$

Vấn đề là thiết lập “điểm dừng” của hàm. Điểm dừng này được cho bởi các giá trị ban đầu của dãy:

$$F_1 = 1$$

$$F_2 = 1$$

Định nghĩa hàm tính số Fibonacci bằng đệ quy.

---

```
>>> def fi(n):
 if n==1:
 return 1
 elif n==2:
 return 1
 else:
 return fi(n-1)+fi(n-2)
```

Thiết lập điểm dừng của hàm tại các trường hợp  $n = 1$  hoặc  $n = 2$ .

Lời gọi đệ quy của hàm.

---

Hàm này đã được định nghĩa đúng và chính xác. Cùng kiểm tra lại.

---

```
>>> fi(20)
10946
>>> fi(25)
121393
>>> fi(30)
1346269
>>> fi(35)
14930352
>>> fi(40)
165580141
>>>
```

---

Chú ý quan trọng: Nếu thử tính số Fibonacci với chỉ số  $> 45$ , chúng ta sẽ thấy thời gian chạy rất lâu. Đó chính là hạn chế của hàm tính số Fibonacci bằng đệ quy, cũng chính là hạn chế của phương pháp đệ quy nói chung. Cách viết hàm đệ quy có ưu điểm là viết nhanh, dễ hiểu nhưng nhược điểm là tốn bộ nhớ, tốc độ chậm. Riêng với chương trình tính các số Fibonacci có thể dùng cách không đệ quy nhanh hơn nhiều.

## 2.2. Hàm tính giai thừa số tự nhiên

Hàm tính giai thừa số tự nhiên:

$$F(n) = 1, 2, \dots, (n-1), n$$

Rõ ràng hàm này có thể được hiệu đê quy với công thức:

$$F(n) = n.F(n-1)$$

với điểm dừng  $F(0) = 1$ .

Hàm số tính giai thừa được viết như sau trên Python.

```
>>> def fac(n):
 if n==0:
 return 1
 else:
 return n*fac(n-1)
```

Kết quả tính thử cho vào giá trị.

[illegible]

Chú ý: Vào những năm 80 của thế kỷ trước khi mới có máy tính PC ra đời, bộ nhớ thấp, tốc độ CPU chậm, khi đó bài toán lập trình tính 100! là một bài toán rất khó.

### 2.3. Hàm đếm countdown

Các hàm đếm được dùng rất nhiều trên thực tế: Đếm thời gian, điểm số, lượt chơi, .... Chúng ta cùng thiết kế một hàm đếm ngược đơn giản nhất: Đếm ngược từ một giá trị số N cho trước và đếm giảm dần đến 1 thì dừng lại.

Hàm đếm ngược được thiết kế dễ dàng như sau với lời gọi tham số giảm dần và điểm dừng khi tham số = 0.

```
>>> def countdown(n):
 if n > 0:
 print(n)
 countdown(n-1)
 else:
 return
```

Kết quả chạy của hàm đếm ngược như sau:

```
>>> countdown(5)
5
4
3
2
1
>>>
```

## 2.4. Hàm tính xuôi và dãy ngược

Ví dụ sau cùng của phần này là bài toán cho trước một chuỗi ký tự hay dãy số, cần tính ngay ra được chuỗi ngược hay dãy ngược lại chuỗi hay dãy này. Ví dụ nếu cho trước, ví dụ:

---

```
s = "abcde"
A = [0, 1, 2, 3, 4, 5]
```

---

Cần viết chương trình để tạo ra chuỗi mới "edcba" và dãy [5, 4, 3, 2, 1, 0].

Các hàm `XauN()` và `DayN()` được thiết kế đơn giản bằng phương pháp đệ quy. Chú ý ở đây chúng ta dùng tính chất tuần tự và slicing của các kiểu dữ liệu này. Các hàm này cũng có thể viết tương tự để áp dụng của kiểu tuple.

Hai hàm `XauN()` và `DayN()` có cấu trúc và cách viết, lời gọi đệ quy hoàn toàn tương tự nhau.

Hàm `XauN(s)` tính chuỗi ngược của s.

---

```
>>> def XauN(s):
 if s == "":
 return ""
 else:
 return s[-1] + XauN(s[:-1])
```

---

Hàm `DayN(A)` tính dãy số ngược của list A.

---

```
>>> def DayN(A):
 if A == []:
 return []
 else:
 return A[len(A)-1:] + DayN(A[:len(A)-1])
```

---

Kết quả thể hiện như sau.

---

```
>>> XauN("abcde")
'edcba'
>>> DayN([0, 1, 2, 3, 4, 5])
[5, 4, 3, 2, 1, 0]
>>>
```

---

Chú ý: Dùng kỹ thuật slicing (vùng chỉ số) có thể dễ dàng thiết lập chuỗi hoặc dãy số ngược như sau. Nhưng trong phần này chúng ta học về đệ quy nên cần các ví dụ minh họa cho kỹ thuật đệ quy.

---

```
>>> s[::-1]
'edcba'
>>> A[::-1]
[5, 4, 3, 2, 1, 0]
```

---

### 3. Một vài ứng dụng đệ quy

#### 3.1. Sắp xếp trộn (merge sort)

Thuật toán sắp xếp trộn được mô tả trong phần này được xây dựng dựa trên kỹ thuật trộn 2 mảng đã sắp xếp đúng thành một mảng tổng hợp cũng được sắp xếp.

Bảng sau mô tả các bước trộn 2 dãy số như vậy.

| Bước | Dãy gốc            | Số tìm đưa vào kq | Kết quả thu được |
|------|--------------------|-------------------|------------------|
| 1    | [1, 7, 10], [2, 5] | 1                 | [1]              |
| 2    | [7, 10], [2, 5]    | 2                 | [1, 2]           |
| 3    | [7, 10], [5]       | 5                 | [1, 2, 5]        |
| 4    | [7, 10]            | [7, 10]           | [1, 2, 5, 7, 10] |

Thủ tục trộn `merge()` có 2 tham số đầu vào là 2 dãy `left`, `right` đều đã được sắp xếp đúng. Từng bước, thủ tục sẽ kiểm tra các số hạng đầu tiên của 2 dãy, tìm ra số nhỏ hơn, đưa số này vào dãy kết quả và xóa số đó khỏi dãy ban đầu. Quá trình trên tiếp tục cho đến khi một trong 2 danh sách ban đầu bằng rỗng. Khi đó sẽ đưa nốt dãy còn lại vào dãy kết quả.

```
def merge(left, right):
 kq = []
 while left != [] and right != []:
 if len(left) > 0 and len(right) > 0:
 if left[0] <= right[0]:
 kq.append(left[0])
 left = left[1:]
 else:
 kq.append(right[0])
 right = right[1:]
 if len(left) == 0:
 kq.extend(right)
 break
 if len(right) == 0:
 kq.extend(left)
 break
 return kq
```

Kiểm tra nếu cả 2 dãy gốc đều không rỗng thì tìm phần tử đầu tiên nhỏ nhất để đưa vào kết quả.

Nếu một trong 2 dãy gốc là rỗng thì đưa dãy còn lại vào kết quả và kết thúc thuật toán.

Bây giờ thủ tục chính `merge_sort()` sẽ tách dãy cần sắp xếp ban đầu thành các dãy nhỏ, dùng kỹ thuật đệ quy để tách các dãy này, sau đó dùng thủ tục `merge()` để nối các dãy nhỏ lại với nhau thành dãy lớn đã sắp xếp.

Thủ tục chính đệ quy `merge_sort()` như sau.

```
def merge_sort(A):
 if len(A) <= 1:
 return A
 left = A[: len(A)// 2]
 right = A[len(A)// 2:]
 sortedleft = merge_sort(left)
 sortedright = merge_sort(right)
 return merge(sortedleft, sortedright)
```



### 3.2. Sắp xếp nhanh (quick sort)

Thuật toán **QuickSort** được thực hiện theo ý tưởng của bài toán tìm kiếm nhị phân: Cần thực hiện nhanh nhất để chia dãy đã cho thành 2 phần, ngăn cách bởi 1 phần tử pivot. Toàn bộ bên trái (màu xám) sẽ  $<$  pivot, khu vực bên phải (màu sẫm) sẽ  $>$  pivot. Sau đó dùng thủ thuật đệ quy để phân chia tiếp tục 2 dãy con trái và phải. Giá trị pivot ban đầu có thể lấy một phần tử bất kỳ.



Như vậy thuật toán QuickSort sẽ bao gồm 2 bước:

- Bước 1 dùng thủ tục Partition để tách dãy làm 2 phần, ngăn cách bởi phần tử pivot.
- Bước 2, thực hiện đệ quy QuickSort cho 2 phần đã được tạo ra bởi bước trên.

Thủ tục sau, Partition(A, p, r) có chức năng tìm ra một chỉ số q nằm giữa (p, r) đóng vai trò pivot. Sau khi thủ tục này thực hiện xong, tất cả các phần tử bên trái q sẽ  $\leq$  pivot, tất cả các phần tử bên phải q sẽ  $\geq$  pivot.

Thủ tục Partition() đơn giản như sau.

---

```
>>> def Partition(A, p, r):
 pivot = A[r]
 i = p-1
 for j in range(p, r):
 if A[j] < pivot:
 i = i+1
 A[i], A[j] = A[j], A[i]
 if A[i+1] > A[r]:
 A[i+1], A[r] = A[r], A[i+1]
 return i+1
```

---

Thủ tục đệ quy chính QuickSort() được thực hiện với 3 tham số là A, p, r như sau. Nhiệm vụ của thủ tục này là sắp xếp các phần tử của dãy A trong khoảng chỉ số từ p đến r. Các bước thực hiện như sau:

- Bước 1: Xác định vị trí pivot của dãy, giả sử đó là q. Tới lúc này dãy A đã được tạm phân thành 2 phần ngăn cách bởi phần tử pivot là q.
- Bước 2: Áp dụng lập trình đệ quy gọi chính lại hàm này cho 2 dãy con là A[p : q-1] và A[q+1 : r].

Chương trình như sau.

---

```
>>> def QuickSort(A, p, r):
 if p < r:
 q = Partition(A, p, r)
 QuickSort(A, p, q-1)
 QuickSort(A, q+1, r)
```

---

Ví dụ thực hiện trên một dãy cụ thể như sau:

```
>>> A = [2, 5, 1, -1, 100, 99, 15, 9, 77, 54, 71, -10, 18]
>>> QuickSort(A, 0, len(A)-1)
>>> A
[-10, -1, 1, 2, 5, 9, 15, 18, 54, 71, 77, 99, 100]
>>>
```

### 3.3. Sinh hoán vị ngẫu nhiên

Cho trước dãy số tự nhiên:

1, 2, 3, ..., n. (1)

Dãy:

$a_1, a_2, \dots, a_n$  (2)

bao gồm các số lấy từ dãy (1) được gọi là một hoán vị của dãy (1).

Bài toán của chúng ta là cho trước số tự nhiên n, cần sinh ra tất cả các hoán vị của dãy (1), kết quả thể hiện trên màn hình.

Cách thiết lập các hoán vị sau được gọi là **Thuật toán thay đổi tối thiểu (minimal change algorithm)**. Thuật toán này được xây dựng dựa trên đệ quy từ các bước trước với n nhỏ hơn. Quan sát bảng sau cách thiết lập các hoán vị với  $n = 1, 2, 3$  để hiểu rõ thuật toán này.

|         |                            |                                                                                                                                                                                                                                                                                                      |
|---------|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $n = 1$ | 1                          | Với $n = 1$ chỉ có đúng một hoán vị.                                                                                                                                                                                                                                                                 |
| $n = 2$ | 2 1    1 2                 | Lấy số 2 chèn vào trước và sau số 1, thu được 2 hoán vị.                                                                                                                                                                                                                                             |
| $n = 3$ | 321 231 213    312 132 123 | Với mỗi hoán vị của bước trước, lấy số 3 lần lượt chèn vào các vị trí từ trái sang phải của hoán vị đó, cuối cùng thu được 6 hoán vị của $n = 3$ . Cụ thể như sau:<br>- Với hoán vị 2 1, chèn số 3 thu được 3 hoán vị: 321 231 213.<br>- Với hoán vị 1 2, chèn số 3 thu được 3 hoán vị: 312 132 123. |

Bộ dữ liệu chính dùng trong thuật toán là một dãy (list) A. Mỗi phần tử của dãy A sẽ là một hoán vị cần thiết lập dưới dạng một list. Như vậy A có khuôn dạng List of List.

Dãy A sẽ được thiết lập đệ quy như sau:

Với  $n = 1$ ,  $A = [1]$

Với  $n = 2$ ,  $A = [[1, 2], [2, 1]]$

Với  $n = 3$ ,  $A = [[3, 2, 1], [2, 3, 1], [2, 1, 3], [3, 1, 2], [1, 3, 2], [1, 2, 3]]$

Đoạn chương trình sau mô tả hàm đệ quy sinh dãy A cần tìm.

```

1. def permutation(n):
2. if n == 1:
3. return [[1]]
4. elif n == 2:
5. return [[1,2], [2,1]]
6. else:
7. new = []
8. A = permutation(n-1)
9. for pe in A:
10. new = new + insertALL(n,pe)
11. return new

```

Với  $n = 1, 2$  cần thiết lập A tường minh.

Lời gọi hàm đệ quy.

Hàm `insertALL(n, C)`, với C là một dãy số, sẽ tạo ra một dãy mới bằng cách chèn số n lần lượt vào dãy C. Kết quả trả lại là một dãy mới. Hàm này được cho trong phần bài tập cuối chương.

Toàn bộ chương trình chi tiết của thuật toán này sẽ có trong các bài tập 3, 4, 5 cuối chương này.



## Câu hỏi - bài tập

1. Hàm  $f(n)$  được định nghĩa như sau:

```

def f(n):
 if n == 0:
 return 1
 else:
 return n + f(n-1)

```

Tính  $f(10)$ .

- a) 48
- b) 50
- c) 56
- d) 94

2. Viết hàm đếm thời gian từ 1 trở đi, mỗi lần đếm là 1 giây. Đồng hồ thời gian sẽ dừng lại sau đúng 2 phút (tức sau 120 giây).

3. Hàm số sau thực hiện công việc gì?

```

1. def insertNum(n, A, k):
2. newlist = []
3. if k == 0:
4. newlist = [n] + A
5. elif k == len(A) + 1:
6. newlist = A + [n]
7. else:
8. newlist = A[:k] + [n] + A[k:]
9. return newlist

```

#### 4. Hàm số sau thực hiện công việc gì:

---

```
1. def insertALL(n, A):
2. newlist = []
3. for k in range(len(A) + 1):
4. newlist.append(insertNum(n,A,k))
5. return newlist
```

---

Chú ý: hàm này sử dụng hàm insertNum() trong bài tập 3.

#### 5. Thuật toán sinh hoán vị ngẫu nhiên **Minimal Change**.

Yêu cầu nhập vào từ bàn phím số tự nhiên  $n$ , thể hiện trên màn hình tất cả các hoán vị của dãy  $[1, 2, \dots, n]$ . Chương trình sau là phần chương trình lõi chính của thuật toán này.

Viết lại chương trình sau và trả lời các câu hỏi bên dưới.

---

```
1. def insertNum(n, A, k):
2. newlist = []
3. if k == 0:
4. newlist = [n] + A
5. elif k == len(A) + 1:
6. newlist = A + [n]
7. else:
8. newlist = A[:k] + [n] + A[k:]
9. return newlist
10. def insertALL(n, A):
11. newlist = []
12. for k in range(len(A) + 1):
13. newlist.append(insertNum(n,A,k))
14. return newlist
15. def permutation(n):
16. if n == 1:
17. return [[1]]
18. elif n == 2:
19. return [[1,2], [2,1]]
20. else:
21. new = []
22. A = permutation(n-1)
23. for pe in A:
24. new = new + insertALL(n,pe)
25. return new
26. A = permutation(4)
27. print(A)
```

---

a) Thực hiện hàm số với  $n = 1, 2, 3$  để xem kết quả và dự đoán chức năng chính của hàm **permutation()**.

b) Hàm **permutation(2)** trả lại list  $[[1,2], [2,1]]$ , em có nhận xét gì về kết quả này.

c) Chứng minh rằng hàm **permutation(n)** là hàm sinh các hoán vị của dãy  $1, 2, 3, \dots, n$ .

d) Kết quả hàm **permutation(n)** trả lại dữ liệu gì?

e) Dựa vào đoạn chương trình trên viết lại chương trình mô tả yêu cầu của bài toán này.

## 6. Giải thích chương trình sau.

```
1. from random import *
2. n = int(input("Nhập số tự nhiên n: "))
3. A = []
4. for i in range(1, n+1):
5. A.append(i)
6. C = []
7. while A != []:
8. k = choice(range(len(A)))
9. a = A[k]
10. C.append(a)
11. A.remove(a)
12. print(C)
```

7. Cho trước số tự nhiên  $n$  nhập từ bàn phím.

Viết chương trình tìm tất cả các khai triển:

$N = x_1 + x_2 + \dots + x_k$ , với  $x_i$  là số tự nhiên.

Chương trình cần liệt kê tất cả các bộ nghiệm:

$(x_1, x_2, \dots, x_k)$

Ví dụ:

|       |                                                  |
|-------|--------------------------------------------------|
| N = 1 | 1 = 1                                            |
| N = 2 | 2 = 1 + 1<br>2 = 2                               |
| N = 3 | 3 = 1 + 1 + 1<br>3 = 1 + 2<br>3 = 2 + 1<br>3 = 3 |

Gợi ý:

Bộ dữ liệu được thiết lập là dãy (list)  $A$  có dạng, chú ý rằng chỉ số sẽ đánh số từ 0.

$A = [x_0, x_1, \dots, x_k]$

Lời giải được thực hiện đệ quy bởi hàm **gpt(A,n,p)** như sau:

Lời gọi ban đầu là **gpt(A,N, 0)**.

$$N = x_1 + x_2 + \dots + \overset{\vdots}{x_p} + \overset{\vdots}{x_{p+1}} + \dots + x_k$$

Lời gọi hàm **gpt(A,n,p)** bắt đầu từ vị trí này. Hàm sẽ được thực hiện đệ quy như sau:

```
for k in range(1, n+1):
 if len(A) > p:
 A = Copy(A,p) # xóa các phần tử sau p
 A.append(k)
 gpt(A,n-k, p+1)
```



Chương trình chính sẽ có dạng đệ quy như sau:

```
1 def gpt(A, n, p):
2 if n == 0:
3 print(A)
4 return
5 else:
6 for k in range(1, n+1):
7 if len(A) > p:
8 A = Copy(A, p)
9 A.append(k)
10 gpt(A, n-k, p+1)
11 return
12 # main chương trình chính
13 n = int(input("Nhập số n: "))
14 A = []
15 gpt(A, n, 0)
```

Hàm này có chức năng  
xóa các phần tử của dãy  
A tính từ vị trí p.

8. Giải quyết bài toán trong bài tập 7 theo một gợi ý khác.

List A được thiết lập một lần từ đầu với độ dài cố định (khá lớn). Ví dụ A được thiết lập ban đầu bao gồm toàn các phần tử 0 như sau:

```
MAX = 10000
A = []
for i in range(MAX):
 A.append(0)
```

Thủ tục đệ quy chính có dạng tương tự như sau:

```
1 def gpt(A, n, p):
2 if n == 0:
3 Show(A)
4 return
5 else:
6 for k in range(1, n+1):
7 if len(A) > p:
8 # xóa các phần tử sau chỉ số p
9 A[p] = k
10 gpt(A, n-k, p+1)
11 return
```

Em hãy hoàn thiện chương trình trên và hoàn thiện yêu cầu của bài tập này.

9. Xét bài toán sau tổng quát hơn bài toán trong các bài tập trên:

Cho trước số N và k số tự nhiên cho trước  $m_1, m_2, \dots, m_k$ .

Cần tìm tất cả các nghiệm nguyên (lớn hơn hoặc bằng 0) của phương trình:

$$N = x_1 m_1 + x_2 m_2 + \dots + x_k m_k \text{ với } x_i \geq 0$$

Chương trình cần liệt kê trên màn hình tất cả các nghiệm của phương trình trên. Nếu không có nghiệm chương trình sẽ thông báo vô nghiệm.

Ví dụ với  $k = 3$ .

$m_1 = 1, m_2 = 2, m_3 = 3, N = 5$  chương trình cần liệt kê các nghiệm như sau:

---

|   |   |   |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 2 | 0 |
| 2 | 0 | 1 |
| 3 | 1 | 0 |
| 5 | 0 | 0 |

---

Giải bài toán trên khi bộ dữ liệu nghiệm được lưu trong các dãy (list) có đúng k phần tử dạng:

$A = [a_1, a_2, \dots, a_k]$ .

10. Mở rộng yêu cầu của bài tập 9. Yêu cầu chương trình thông báo tổng số nghiệm tìm được trước khi thể hiện tất cả các nghiệm trên màn hình.

Ví dụ đầu ra của chương trình như sau.

---

Tổng số nghiệm: 5

|   |   |   |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 2 | 0 |
| 2 | 0 | 1 |
| 3 | 1 | 0 |
| 5 | 0 | 0 |

---

11. Viết các hàm số sau sử dụng lập trình đệ quy.

a) Hàm **S(n)** là tổng các số tự nhiên  $\leq n$ .

$S(n) = 1 + 2 + \dots + n$ .

b) Hàm **SEven(n)** là tổng các số tự nhiên chẵn  $\leq n$ .

c) Hàm **SOdd(n)** là tổng các số tự nhiên lẻ  $\leq n$ .

12. Hàm sau đây thực hiện công việc gì?

---

```
1. def Rev(A):
2. if A == []:
3. return []
4. kqLast = Rev(A[1:])
5. item = A[0]
6. return kqLast + item
```

---

13. Dựa trên bài tập 12, viết hàm tương tự nhưng áp dụng cho chuỗi ký tự.

14. Viết các hàm số sau sử dụng lập trình đệ quy.

a) Hàm **Exp(x, n)** tính lũy thừa  $x^n$  với x là số thực bất kỳ, n là số tự nhiên.

b) Hàm **FacEven(n)** tính tích các số tự nhiên chẵn  $\leq n$ .

c) Hàm **FacOdd(n)** tính tích các số tự nhiên lẻ  $\leq n$ .

15. Viết chương trình tính các tổng sau bằng phương pháp đệ quy.

a)  $S = 1 + 2 + \dots + n$

b)  $S = 1^3 + 2^3 + \dots + n^3$

$$c) S = \frac{1}{1.2} + \frac{1}{2.3} + \dots + \frac{1}{n(n+1)}$$

$$d) S = \frac{1}{1} + \frac{1}{1.2} + \frac{1}{1.2.3} + \dots + \frac{1}{n!}$$

16. Cho đoạn chương trình sau:

---

```

1. def LenNum(str):
2. if str == "":
3. return 0
4. elif len(str) == 1:
5. if ord(str[0]) >= 48 and ord(str[0]) <= 57:
6. return 1
7. else:
8. return 0
9. else:
10. if ord(str[0]) >= 48 and ord(str[0]) <= 57:
11. return 1 + LenNum(str[1 :])
12. else:
13. return LenNum(str[1 :])

```

---

a) Chương trình trên có sử dụng đệ quy hay không?

b) Nêu ý nghĩa của hàm số LenNum(str) trong chương trình trên.

c) Chương trình trên có một vài đoạn code chưa tối ưu, bạn có thể viết rút gọn lại được không?

17. Viết các chương trình sau đều yêu cầu nhập từ bàn phím một xâu ký tự và tính:

a) Tính số các ký tự là số trong xâu (các ký tự 0, 1, 2, ..., 9).

b) Tính số các ký tự là chữ cái (chữ hoa hoặc thường) trong xâu.

c) Tính số các ký tự không là số và chữ trong xâu.

Yêu cầu viết các chương trình trên bằng kỹ thuật đệ quy.

18. Viết phương trình tìm tất cả các nghiệm của phương trình:

$$N = x_1 + x_2 + x_3$$

với điều kiện  $0 \leq N \leq 30, 0 \leq x_1, x_2, x_3 \leq 9$ .

19. Tổng quát bài tập 18, viết chương trình tìm tất cả các nghiệm của phương trình:

$$N = x_1 + x_2 + \dots + x_m$$

với điều kiện:

$$N \geq 0, 0 \leq x_1, x_2, x_3, \dots, x_m \leq 9.$$

20. Viết chương trình giải phương trình tổng quát của bài tập 19, hạn chế các nghiệm  $x_i$  theo giới hạn cho trước:

$$0 \leq x_i \leq C_i, \text{ với } C_1, C_2, \dots, C_m \text{ là các hằng số cho trước.}$$

## Chương 13. Kiểu dữ liệu từ điển

### Mục tiêu:

- Kiểu dữ liệu từ điển trong Python.
- Các phương thức của dữ liệu từ điển.
- Một số ứng dụng với dữ liệu từ điển.



### Nội dung chính

Dữ liệu kiểu list và tuple mà chúng ta đã biết là các cấu trúc dữ liệu có chung đặc điểm sau:

- Là kiểu thùng dữ liệu (container), không phải là kiểu dữ liệu đơn giản.
- Là dữ liệu tuần tự (iterable): Các phần tử của list hay tuple đều được sắp thứ tự, đánh chỉ số từ 0, có thể truy cập theo từng phần tử hoặc theo slicing - một vùng liên tục các phần tử. Vậy list và tuple là các kiểu dữ liệu cổ điển dạng dãy, mảng số.

Dữ liệu từ điển là một dạng dữ liệu khác, không tuần tự, không là dãy và có cấu trúc dưới dạng các cặp <khóa>-<giá trị>. Mỗi dữ liệu từ điển sẽ bao gồm nhiều khóa, mỗi khóa sẽ tương ứng với giá trị riêng của mình.

Có thể hiểu đơn giản dữ liệu từ điển như một bản ghi của bảng trong mô hình CSDL quan hệ. Trong mô hình này, tên của các cột chính là khóa, giá trị tương ứng của bản ghi chính là giá trị tương ứng của khóa.

Cấu trúc tổng quát của từ điển có dạng là một dãy các bộ <key-value> (khóa - giá trị), được viết bên trong dấu {}.

---

```
{<key1> : <value1>, <key2> : <value2>, ..., <keyN> : <valueN>}
```

---

Để tạo một đối tượng từ điển rỗng chúng ta dùng ký hiệu {}, ví dụ.

---

```
>>> d = {}
>>> type(d)
<class 'dict'>
```

“dict” là tên của kiểu dữ liệu từ điển.

---

Chú ý: có thể dùng lệnh sau để tạo một từ điển rỗng:

---

```
>>> d = dict()
```

---

Dữ liệu từ điển có thể được gán trực tiếp từ dòng lệnh như lệnh sau.

---

```
>>> hs = {"Tên": "Nguyễn Văn Minh", "Lớp": "6A", "Địa chỉ": "Phố Tràng Thi"}
>>> hs
{'Tên': 'Nguyễn Văn Minh', 'Lớp': '6A', 'Địa chỉ': 'Phố Tràng Thi'}
```

---

Trong mô hình dữ liệu trên, các khóa sẽ bao gồm: “Tên”, “Lớp” và “Địa chỉ”. Các giá trị tương ứng của khóa là “Nguyễn Văn Minh”, “6A”, “Phố Tràng Thi”.

Có thể quan sát hoặc in toàn bộ dữ liệu từ điển, hoặc truy xuất giá trị của một khóa tương ứng.

Xem ví dụ sau.

```
>>> print(hs)
{'Tên': 'Nguyễn Văn Minh', 'Lớp': '6A', 'Địa chỉ': 'Phố Tràng
Thị'}
>>> hs["Tên"]
'Nguyễn Văn Minh'
>>>
```

Cách truy cập một khóa bất kỳ của dữ liệu.

Cách truy cập giá trị của một khóa như sau: <tên từ điển>[<khóa>].

Ví dụ sau là một cách thiết lập dữ liệu từ điển theo cách khác, theo từng khóa. Bắt đầu từ dữ liệu rỗng, chúng ta sẽ nhập dần các dữ liệu cho từng khóa.

Dữ liệu điểm trung bình các môn của một học sinh.

```
>>> student = {}
>>> student["Toán"] = 9.5
>>> student["Hóa"] = 7.1
>>> student["Văn"] = 7.9
>>> student["Sinh"] = 10
>>> student
{'Toán': 9.5, 'Hóa': 7.1, 'Văn': 7.9, 'Sinh': 10}
>>>
```

Chú ý quan trọng:

- Phần <key> của từ điển có thể có kiểu dữ liệu số, xâu, tuple, nhưng không được phép là list.
- Phần <value> của từ điển có thể có kiểu dữ liệu bất kỳ, là số, xâu, tuple, list, thậm chí có thể là từ điển hay tập hợp<sup>7</sup>.

Các phương thức của dữ liệu từ điển.

```
>>> dir(student)
['_class_', '__contains__', '__delattr__', '__delitem__',
'_dir_', '__doc__', '__eq__', '__format__', '__ge__',
'_getattribute__', '__getitem__', '__gt__', '__hash__',
'_init_', '__init_subclass__', '__iter__', '__le__', '__len__',
'_lt_', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'_repr_', '__setattr__', '__setitem__', '__sizeof__',
'_str_', '__subclasshook__', 'clear', 'copy', 'fromkeys',
'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update',
'values']
```

Chúng ta sẽ cùng xem xét kỹ hơn các phương thức in đậm trong danh sách trên.

## 1. Các lệnh keys() và values()

Dữ liệu của từ điển là các cặp khóa - giá trị. Nếu biết <khóa> thì dễ dàng tính được các giá trị thông qua cách truy cập trực tiếp <dict>[<key>].

Ví dụ với dữ liệu student thì có thể lấy điểm số từng môn học nếu biết tên môn.

<sup>7</sup> Khái niệm dữ liệu set (tập hợp) sẽ được học trong bài sau.



```
>>> student["Toán"]
9.5
>>>
```

Để biết được tất cả các tên khóa chúng ta sử dụng cấu trúc duyệt dữ liệu từ điển. Có thể làm như cách sau.

```
>>> for mon in student:
 print(mon)
Toán
Hóa
Văn
Sinh
>>>
```

Nhưng cách hợp lý nhất là sử dụng phương thức `keys()`. Phương thức này cho phép chúng ta duyệt trên danh sách các khóa của dữ liệu thông qua lệnh **for**. Ví dụ như sau.

```
>>> for mon in student.keys():
 print(mon)
Toán
Hóa
Văn
Sinh
>>>
```

Tương tự muốn duyệt trực tiếp các giá trị của từ điển, chúng ta dùng phương thức `values()`. Tương tự như với cách duyệt khóa, lệnh sau sẽ duyệt trên danh sách các giá trị của từ điển.

```
>>> for diem in student.values():
 print(diem)
9.5
7.1
7.9
10
>>>
```

Vì từ các khóa <key> có thể tính được giá trị <value>, nên nếu duyệt khóa bằng phương thức `keys()`, chúng ta có thể lấy được cả các giá trị của từ điển.

```
for key in student.keys():
 print("Điểm môn {:6} là: {}".format(key, student[key]))
Điểm môn Toán là: 9.5
Điểm môn Hóa là: 7.1
Điểm môn Văn là: 7.9
Điểm môn Sinh là: 10
>>>
```

Đoạn lệnh sau cho phép hiển thị môn học và điểm của học sinh, nhưng đã được sắp xếp trước theo tên gọi của key.

```
>>> for key in sorted(student):
 print("Điểm trung bình môn {:6} là: {}".format(key,
student[key]))
Điểm trung bình môn Hóa là: 7.1
Điểm trung bình môn Sinh là: 10
Điểm trung bình môn Toán là: 9.5
Điểm trung bình môn Văn là: 7.9
```

## 2. Lệnh items()

Phương thức items() có ý nghĩa gần giống keys() và values() nhưng cho phép duyệt đồng thời vừa khóa và giá trị. Lệnh này rất hữu ích. Cùng xem ví dụ sau để hiểu ý nghĩa của phương thức items().

```
>>> for mon, diem in student.items():
 print(mon, diem)
Toán 9.5
Hóa 7.1
Văn 7.9
Sinh 10
```

## 3. Lệnh update()

Lệnh update() có chức năng cập nhật thông tin cho dữ liệu từ điển hiện thời. Cú pháp của lệnh ở dạng đơn giản như sau.

```
<dict Object>.update(T2)
```

Trong đó <T2> là một đối tượng từ điển khác nhưng có các giá trị khóa giống hoặc là một tập hợp con của khóa của đối tượng chính <dict Object>.

Khi thực hiện lệnh update() trên, thực chất Python sẽ thực hiện lệnh sau.

```
for key in T2:
 <dict Object>[key] = T2[key]
```

Xét ví dụ với dữ liệu student đã có. Chúng ta tạo thêm từ điển T là điểm tổng kết mới môn Toán và Văn của học sinh này. Sau khi có T có thể cập nhật thông tin điểm từ T vào student.

Chương trình minh họa như sau.

```
>>> student
{ 'Toán': 9.5, 'Hóa': 7.1, 'Văn': 7.9, 'Sinh': 10}
>>> T = { "Toán": 10, "Văn" : 9.5}
>>> student.update(T)
>>> student
{ 'Toán': 10, 'Hóa': 7.1, 'Văn': 9.5, 'Sinh': 10}
```

## 4. Lệnh get()

Chúng ta đã biết, dữ liệu từ điển sẽ được truy cập giá trị trực tiếp từ khóa (key). Nếu không biết khóa hoặc nhập khóa sai thì chương trình sẽ báo lỗi.

Trong ví dụ sau, chúng ta cố gắng truy cập giá trị điểm trung bình của môn Tin học, nhưng key này không có trong dữ liệu nên xảy ra lỗi.

```
>>> hs["Tin học"]
Traceback (most recent call last):
 File "<pyshell#68>", line 1, in <module>
 hs["Tin học"]
KeyError: 'Tin học'
```

Lỗi nếu truy xuất trực tiếp vào key không có trong dữ liệu.

Hàm get() là một cách khác để truy cập giá trị của dữ liệu từ điển. Ví dụ sau cho thấy, hàm get() được gọi trực tiếp với tham số là khóa thì sẽ có tác dụng như truy cập trực tiếp mà ta đã biết.

```
>>> student.get("Toán")
9.5
```

Tuy nhiên sự khác biệt sẽ xảy ra nếu nhập khóa sai.

Đoạn chương trình sau cho chúng ta thấy, nếu nhập khóa sai thì Python sẽ không báo lỗi mà trả lại giá trị None.

```
>>> student.get("Tin")
>>> type(student.get("Tin"))
<class 'NoneType'>
```

Hay tổng quát hơn, hàm get() có cú pháp như sau.

```
<dict>.get(key [, default])
```

Nếu key nhập đúng thì hàm sẽ trả lại đúng giá trị của từ điển tương ứng. Nhưng nếu khóa không đúng thì hàm get() sẽ trả lại giá trị default nếu có ghi trong tham số của lệnh.

Ví dụ minh họa trong lệnh sau khi môn Tin học không có trong danh sách các từ khóa.

```
>>> student.get("Tin học", "Chưa có điểm trung bình")
'Chưa có điểm trung bình'
>>>
```

Bây giờ giả sử chúng ta có một dãy danh sách các môn học.

```
daymon = ["Toán", "Tin", "Văn", "Lý", "Hóa", "Sinh"]
```

Cần liệt kê theo các môn học này điểm trung bình tương ứng của học sinh.

Chương trình đơn giản sau cho chúng ta lời giải nếu dùng lệnh get().

```
>>> student = {'Toán': 9.5, 'Hóa': 7.1, 'Văn': 7.9, 'Sinh': 10}
daymon = ["Toán", "Tin", "Văn", "Lý", "Hóa", "Sinh"]
for mon in daymon:
 print("Môn {:6} điểm trung bình: {} ".format(mon,
student.get(mon, "Chưa có điểm trung bình")))
```

Kết quả của chương trình trên sẽ như sau.

```
Môn Toán điểm trung bình: 9.5
Môn Tin điểm trung bình: Chưa có điểm trung bình
Môn Văn điểm trung bình: 7.9
Môn Lý điểm trung bình: Chưa có điểm trung bình
Môn Hóa điểm trung bình: 7.1
Môn Sinh điểm trung bình: 10
```

## 5. Xóa khóa và dữ liệu của từ điển

Một khóa của từ điển có thể xóa bằng lệnh del.

```
del <dict Object>[<key>]
```

Ví dụ:

```
D = {"Hà": 10, "Bình": 9.5}
>>> del D["Hà"]
>>> D
{'Bình': 9.5}
```

Muốn xóa toàn bộ tất cả dữ liệu của từ điển chúng ta dùng hàm clear().

```
<dict Object>.clear()
```

## 6. Một số ứng dụng đơn giản của dữ liệu từ điển

Chúng ta hay gặp bài toán thực tế cần đếm số lượng các hạng mục hay đồ vật trong một kho hàng hóa. Hay bài toán đếm số lần lặp của các từ đơn trong một văn bản bất kỳ.

Chúng ta sẽ cùng nhau làm bài toán số 2 sử dụng kỹ thuật từ điển của Python.

Giả sử cho trước một văn bản bất kỳ. Ở đây văn bản sẽ được cho bởi xâu ký tự s. Muốn nhập s trên nhiều dòng chúng ta dùng 3 dấu nhảy đúp ở 2 đầu để có thể nhập xâu s trên nhiều dòng lệnh.

Ví dụ xâu s như sau.

```
>>> s = """This may seem extremely limiting, but a single bit is
enough to represent whether the answer to a question is yes or no
or a single bit can be used to tell us whether a logical statement
evaluates to either true or false"""
>>> A = s.split()
```

Danh sách (list) A được thiết lập là dãy các từ đơn lấy từ xâu s. A được tính bằng lệnh split() của xâu s. Kết quả A sẽ như sau.

```
>>> A
['This', 'may', 'seem', 'extremely', 'limiting,', 'but', 'a',
'single', 'bit', 'is', 'enough', 'to', 'represent', 'whether',
'the', 'answer', 'to', 'a', 'question', 'is', 'yes', 'or', 'no',
'or', 'a', 'single', 'bit', 'can', 'be', 'used', 'to', 'tell',
'us', 'whether', 'a', 'logical', 'statement', 'evaluates', 'to',
'either', 'true', 'or', 'false']
>>>
```

Chương trình chính dùng để tính và đếm số lần lặp lại của các từ đơn trong xâu s sẽ như sau.

```
1. dic = {}
2. for w in A:
3. dic[w] = dic.get(w,0) + 1
4. for key in dic:
5. print(key, dic[key])
```

- Dòng lệnh 1 sẽ tạo ra một biến dữ liệu kiểu từ điển **dic** là rỗng. Từ điển này sẽ có khóa là các từ có trong văn bản trên, giá trị tương ứng là số lần lặp lại của khóa đó trong văn bản đang xét.

- Dòng lệnh 2 là lệnh chính dùng để kiến tạo từ điển **dic**. Với mỗi từ đơn lấy từ danh sách A, lệnh `dic[w] = dic.get(w, 0) + 1` có ý nghĩa: nếu khóa w đã có thì giá trị của khóa này sẽ tăng lên 1 đơn vị, nếu khóa này chưa có thì sẽ bổ sung vào danh sách khóa với giá trị 0. Hàm `get(w, 0)` thực hiện đúng yêu cầu đó của chúng ta.

- Dòng lệnh 4 là lệnh duyệt theo từ điển **dic** và in các khóa và giá trị của khóa/

Kết quả chương trình đơn giản trên như sau.

```
This 1
may 1
seem 1
extremely 1
limiting, 1
but 1
a 4
```



single 2  
bit 2  
is 2  
enough 1  
to 4  
represent 1  
whether 2  
the 1  
answer 1  
question 1  
yes 1  
or 3  
no 1  
can 1  
be 1  
used 1  
tell 1  
us 1  
logical 1  
statement 1  
evaluates 1  
either 1  
true 1  
false 1

---



## Câu hỏi - bài tập

1. Giả sử `d` là một biến dữ liệu kiểu từ điển cho trước. Khi đó lệnh:

---

```
d[1] = "Hà"
```

---

sẽ thực hiện công việc gì?

- a) Tạo thêm một bản ghi (cặp key - value) mới `<1 : "Hà">` và đưa vào phần tử đầu tiên của từ điển.
- b) Tạo thêm một bản ghi (cặp key - value) mới `<1 : "Hà">` và đưa vào phần tử cuối cùng của từ điển.
- c) Nếu trong từ điển đã có key (khóa) = 1 rồi thì cập nhật giá trị mới "Hà" vào khóa này, ngược lại thì tạo thêm một bản ghi (cặp key - value) mới `<1 : "Hà">` và đưa vào phần tử cuối cùng của từ điển.
- d) Lệnh báo lỗi.

2. Giả sử từ điển `d` được định nghĩa như sau:

```
d = {1: 'Bình', 3: 'Thuận', 2: 'Quang'}
```

Khi đó lệnh:

---

```
list(d)
```

---

trả lại kết quả gì?

- a) Dãy (list) các cặp key-value của từ điển.
- b) Dãy (list) các key của từ điển.



c) Dãy (list) các value của từ điển.

3. Giả sử từ điển d được định nghĩa như sau:

d = {1: 'Bình', 3: 'Thuận', 2: 'Quang'}

Khi đó lệnh:

---

```
print(d["Quang"])
```

---

trả lại kết quả gì?

a) 2

b) "Quang"

c) [2 : "Quang"]

d) Chương trình báo lỗi.

4. Giả sử từ điển d được định nghĩa như sau:

d = {1: 'Bình', 3: 'Thuận', 2: 'Quang'}

Khi đó lệnh:

---

```
d.get(0)
```

---

trả lại kết quả gì?

a) Chương trình báo lỗi.

b) None

c) Trả lại một kết quả khác với 2 kết quả trên.

5. Giả sử từ điển d được định nghĩa như sau:

d = {1: 'Bình', 3: 'Thuận', 2: 'Quang'}

Khi đó lệnh:

---

```
d.get(0, "Hà")
```

---

trả lại kết quả gì?

a) Chương trình báo lỗi.

b) None

c) "Hà"

d) Một kết quả khác.

6. Viết chương trình tạo và nhập dữ liệu từ điển từ một text file như sau:

a) Bộ dữ liệu từ điển bao gồm Key là họ tên học sinh, Value là điểm số.

b) Tập dữ liệu có dạng sau:

- Dòng đầu tiên ghi số N là số học sinh trong lớp.

- N dòng tiếp theo là dữ liệu điểm học sinh, mỗi dòng bao gồm Họ tên học sinh và giá trị điểm số, cách nhau bởi dấu phẩy.

Ví dụ khuôn dạng một tệp như vậy:

---

5

Nguyen Hoa Binh, 9.5

Bui Viet Ha, 8

Pham Quang Vinh, 7.5

Nguyen Thi Hoa, 10

Truong Van Quang, 8.5

---

7. Giả sử từ điển dic được định nghĩa như sau:

dic = {'b': 2, 'c': 3, 'a': 1}

Khi đó đoạn chương trình sau cho ta kết quả gì?

---

```
for x in sorted(dic):
 print(d[x], end = " ")
```

---

a) a b c

b) 0 1 2

c) Chương trình báo lỗi.

8. Giả sử từ điển dic được định nghĩa như sau:

dic = {'b': 2, 'c': 3, 'a': 1}

Khi đó đoạn chương trình sau cho ta kết quả gì?

---

```
for x in sorted(dic):
 print(x, end = " ")
```

---

a) a b c

b) 0 1 2

c) Chương trình báo lỗi.

9. Cho đoạn chương trình sau:

---

```
dic = {1 : 'a', 2 : 'b', 4 : "d", 10 : "k"}
b = 0 in dic
```

---

b sẽ nhận giá trị nào?

a) True

b) False

c) Giá trị khác với True và False.

d) Chương trình báo lỗi.

10. Cho đoạn chương trình sau:

---

```
dic = {1 : 'a', 2 : 'b', 4 : "d", 10 : "k"}
b = 'a' in dic
```

---

b sẽ nhận giá trị nào?

a) True

b) False

c) Giá trị khác với True và False.

d) Chương trình báo lỗi.

11. Giả sử chúng ta đã có một dữ liệu từ điển thông tin điểm học sinh bao gồm cặp <key>-<value> là <Họ tên> - <Điểm số>. Ví dụ bộ dữ liệu gốc đã có:

```
dic = {"Bùi Việt Hà" : 8, "Nguyễn Thanh Bình" : 9.5}
```

Viết một chương trình có chức năng cập nhật điểm số học sinh vào bộ dữ liệu trên. Chương trình thực hiện như sau:

Chương trình yêu cầu nhập lần lượt 2 tham số:

1) Nhập họ tên học sinh.

2) Nhập điểm số.

Các thông tin trên sẽ được cập nhật vào bộ dữ liệu, hoặc tạo mới một bản ghi, hoặc cập nhật điểm của học sinh đã có trong danh sách. Chương trình sẽ đưa ra thông báo như sau:

Đã tạo mới một học sinh và nhập điểm.

hoặc

Đã cập nhật thông tin điểm của học sinh này.

12. Từ điển D được định nghĩa bao gồm Khóa là tên các bạn học sinh trong lớp, Value là cặp 2 giá trị số là điểm trung bình Văn và Toán. Ví dụ một bộ dữ liệu từ điển như vậy. Bộ dữ liệu điểm được ghi trong một list.

---

```
D = {"Hà" : [7, 9.5], "Bình" : [10, 8], "Hoa" : [5.5, 9]}
```

---

Viết chương trình nhập bộ dữ liệu từ điển trên như sau:

- Đầu tiên yêu cầu người dùng nhập tên học sinh.

- Tiếp theo yêu cầu nhập điểm Văn và Toán trên một dòng bởi 2 số, cách nhau bởi dấu phẩy.

- Tiếp theo thông báo đã cập nhật xong. Có 2 loại thông báo như sau:

- Thông báo loại 1: "Đã nhập điểm cho học sinh mới".

- Thông báo loại 2: "Đã cập nhật lại điểm cho học sinh trong danh sách".

13. Cho trước bộ dữ liệu điểm Văn, Toán của các bạn trong lớp em như trong bài tập 12, ví dụ một bộ dữ liệu như vậy:

---

```
D = {"Hà" : [7, 9.5], "Bình" : [10, 8], "Hoa" : [5.5, 9]}
```

---

Viết chương trình tính điểm trung bình của học sinh lớp em và thể hiện trên màn hình như sau:

---

Điểm trung bình:

Hà: 8.25

Bình: 9.0

Hoa: 7.25

---

#### 14. Cho đoạn chương trình sau:

```
>>>D = {"Hà" : [7, 9.5], "Bình" : [10, 8], "An" : [6, 9]}
>>> Dic = sorted(D)
```

Khi đó Dic có kết quả gì?

a) Dic = {"Hà" : [7, 9.5], "Bình" : [10, 8], "An" : [6, 9]}

b) Dic = {"An" : [6, 9], "Hà" : [7, 9.5], "Bình" : [10, 8]}

c) Dic = {"An", "Hà", "Bình"}

d) Chương trình báo lỗi.

15. Cho trước 3 bộ dữ liệu từ điển các bạn học sinh lớp em. Cả 3 bộ dữ liệu này đều có khuôn dạng giống nhau với bộ <key - value> là <Tên học sinh - Điểm số>, nhưng đó là 3 bộ điểm số các môn Toán, Văn, Ngoại ngữ.

Viết chương trình cho trước 3 bộ từ điển trên là DToan, DVan, DNgoaingu. Cần tạo ra một bộ dữ liệu mới có tên DTB sẽ tính điểm trung bình của 3 môn trên.

Tuy nhiên do dữ liệu của từng môn có thể thiếu nên danh sách tên học sinh của các môn trên có thể thiếu, do vậy chương trình yêu cầu chỉ tính điểm trung bình cho các bạn học sinh có đủ điểm cả 3 môn. Chương trình sẽ bỏ qua không tính điểm trung bình của các bạn học sinh thiếu điểm của một trong 3 môn trên.

16. Hoàn thiện chương trình ứng dụng đã có trong bài học như sau: Cho trước tệp văn bản Vanban.txt lưu trữ đoạn văn bản (có thể lớn bất kỳ). Chương trình sẽ đọc dữ liệu từ tệp văn bản này và in ra màn hình tần suất xuất hiện các từ đơn trong văn bản.

17. Dữ liệu đầu vào như bài 16. Chương trình yêu cầu nhập từ bàn phím số tự nhiên n và in ra màn hình n từ có tần suất xuất hiện lớn nhất của văn bản. Hiển thị các từ và số lần xuất hiện của từ trong văn bản.

18. Dữ liệu đầu vào như bài 16. Cho trước dãy (list) các từ khóa nào đó, Gọi dãy này là KeyWords. Viết chương trình thiết lập một từ điển DIC với tập các key lấy từ KeyWords và giá trị là tần suất xuất hiện của các từ này trong văn bản đã cho.

19. Cho trước từ điển D. Viết chương trình chuyển đổi dữ liệu của từ điển D sang dữ liệu dạng list với đầy đủ cả khóa và giá trị.

Ví dụ nếu D có dạng: {"Hà": 8, "Bình": 9.5, "Hoa": 10} thì list thu được sẽ phải là:

[["Hà", 8], ["Bình", 9.5], ["Hoa", 10]].

20. Cho trước một số tệp văn bản. Các tệp này có thể được bổ sung vào chương trình bất cứ lúc nào. Viết chương trình dùng để khảo sát tần suất xuất hiện các từ khóa lặp nhiều nhất. Chương trình có các chức năng sau:

1) Nạp thêm các văn bản từ tệp vào bộ nhớ. Mỗi lần nạp văn bản, từ điển sinh bởi chương trình sẽ được cập nhật (xem chức năng 2).

2) Thiết lập một từ điển với 10 từ khóa xuất hiện nhiều nhất trong các văn bản đã khảo sát (đã nhập vào từ chức năng 1).

3) In danh sách 10 từ khóa hot nhất cùng với tần suất xuất hiện các từ này trong các văn bản đã khảo sát.

## Chương 14. Kiểu dữ liệu tập hợp

### Mục tiêu:

- Khái niệm tập hợp, kiểu tập hợp trong Python.
- Các phép toán trên kiểu dữ liệu tập hợp.
- Một số ứng dụng đơn giản của tập hợp.



### Nội dung chính

Tập hợp là một khái niệm cơ bản của toán học. Tập hợp được hiểu là một nhóm các phần tử không giống nhau được kết hợp lại. Python cũng hỗ trợ kiểu dữ liệu tập hợp. Các phần tử của tập hợp trong Python có thể có kiểu dữ liệu cơ bản bất kỳ như số, logic, xâu ký tự, tuple là các kiểu dữ liệu dạng không thay đổi (immutability). Tập hợp không thể chứa phần tử là list, dict hay set (là các kiểu dữ liệu container nhưng mutability - có thể thay đổi).

Chú ý: Các phần tử của dữ liệu tập hợp không có chỉ số thứ tự như list, tuple hay string, tuy nhiên vẫn có thể thực hiện lệnh duyệt **for** và toán tử **in** trên tập hợp.

### 1. Khởi tạo tập hợp

#### a. Khởi tạo dữ liệu tập hợp

Tập hợp được khởi tạo và viết thành dãy các phần tử nằm trong ngoặc nhọn.

```
>>> A = {1, 2, 3, "One"}
>>> type(A)
<class 'set'>
```

Nhắc lại các phần tử của tập hợp có thể là số, logic, xâu ký tự hay tuple, nhưng không thể là list, dict hay set<sup>8</sup>.

Trong ví dụ sau khi khai báo có phần tử là list trong tập hợp sẽ báo lỗi.

```
>>> A = {1, 2, 3, [1, 2]}
Traceback (most recent call last):
 File "<pyshell#22>", line 1, in <module>
 A = {1, 2, 3, [1, 2]}
TypeError: unhashable type: 'list'
>>>
```

Các phần tử của tập hợp không thể trùng nhau. Khi khai báo nếu chúng ta viết các phần tử giống nhau thì Python sẽ tự động loại ra khỏi định nghĩa của tập hợp.

Ví dụ sau khi nhập chúng ta viết có nhiều phần tử giống nhau, nhưng Python sẽ tự động loại bỏ.

```
>>> D = {1, 1, 2, 2}
>>> D
{1, 2}
```

<sup>8</sup> Python hỗ trợ kiểu *frozenset* là kiểu tương tự như tập hợp nhưng là immutable, hay còn gọi là hashable. Kiểu dữ liệu này cần khởi tạo bằng lệnh `frozenset()` và sau đó các thao tác hoàn toàn tương tự như tập hợp bình thường. Kiến thức này vượt quá giới hạn của cuốn sách nên không trình bày ở đây.



Chương trình nhỏ sau đây là một “mẹo” có thể lọc các phần tử trùng nhau trong list. Cách làm theo 2 bước, bước 1 chuyển list này sang tập hợp, bước 2 chuyển ngược lại từ tập hợp về list.

```
>>> a = [1, 2, 2, 3, 3, 3, 4, 4]
>>> C = set(a)
>>> C
{1, 2, 3, 4}
>>> a = list(C)
>>> a
[1, 2, 3, 4]
>>>
```

Chú ý: Để tạo tập hợp rỗng chúng ta dùng lệnh `set()` chứ không được dùng `{}`, ví dụ.

```
>>> A = set()
>>> type(A)
<class 'set'>
>>> print(A)
set()
>>> A.add(1)
>>> A
{1}
>>>
```

Ký hiệu `{}` để chỉ dữ liệu từ điển.

```
>>> d = {}
>>> type(d)
<class 'dict'>
>>>
```

### ***b. Tạo tập hợp bằng chức năng set comprehension***

Hàm `set()` có thể biến tất cả các dữ liệu dạng container về kiểu tập hợp. Ví dụ sau minh họa lệnh `set()` có thể chuyển đổi xâu ký tự về tập hợp các ký tự tạo nên xâu này.

```
>>> s = "abcdef"
>>> S = set(s)
>>> S
{'e', 'f', 'a', 'd', 'b', 'c'}
```

Chú ý: lệnh `set()` trên dữ liệu từ điển sẽ chỉ tạo tập hợp từ các khóa.

```
>>> d = {1 : "Bình", 2: "Quang", 3 : "Vinh"}
>>> set(d)
{1, 2, 3}
>>>
```

Một cách tạo tập hợp nữa rất hữu ích là sử dụng lệnh mở rộng của `set()`, gọi là `set comprehension`.

Quan sát ví dụ sau là một lệnh set comprehension. S được kiến tạo là các mã ký tự ASCII của các ký tự có trong chuỗi s.

```
>>> S = {ord(ch) for ch in s} # cách viết lệnh set mở rộng
>>> S
{97, 98, 99, 100, 101, 102}
>>>
```

Cú pháp tổng quát của set comprehension.

Biểu thức bất kỳ trên đối số <x>      Đối số <x>, có thể dùng ký tự bất kỳ.      Điều kiện lọc phần tử cho tập hợp.

---

<set> = {<biểu thức> for <x> in <sequence> if <điều kiện>}

Biến dữ liệu dạng container có hỗ trợ lệnh for.

Ví dụ 1: Cho trước dãy số A. Thiết lập tập hợp S là các phần tử khác 0 lấy từ A.

```
>>> A = [1, 0, 0, -3, 8, 100, 0, 200, 13, 37, 45]
>>> S = {x for x in A if x}
>>> S
{1, 100, 37, 8, 200, 13, 45, -3}
>>>
```

Điều kiện if x tương đương với x != 0.

Ví dụ 2: Cho 2 dãy số A, B. Cần thiết lập tập hợp các số từ dãy A nhưng không nằm trong dãy B.

```
>>> A = [1, 2, 4, 5, 7, 8, 10, 12, 20, 34, 46, 50, 100]
>>> B = [0, 1, 5, 10, 15, 20, 30, 50]
>>> C = {x for x in A if x not in B}
>>> C
{2, 34, 4, 100, 7, 8, 12, 46}
>>>
```

Điều kiện x không nằm trong dãy B.

Ví dụ 3: Cho dãy số A, thiết lập tập hợp các ký tự từ bảng mã ASCII tương ứng với mã hóa nằm trong A.

```
>>> A = [0, 34, 1, 56, 78, 63, 12, 10, 200, 100, 97, 88, 125, 500, 1001]
>>> C = {chr(x) for x in A if x > 31 and x < 127}
>>> C
{' ', 'N', 'a', '?', '8', '}', 'd', 'X'}
>>>
```

Các phương thức của kiểu dữ liệu set.

```
>>> dir(set)
['_and_', '__class__', '__contains__', '__delattr__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__gt__', '__hash__', '__iand__', '__init__',
 '__init_subclass__', '__ior__', '__isub__', '__iter__',
 '__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__',
 '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__',
 '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__',
 '__str__', '__sub__', '__subclasshook__', '__xor__', 'add',
 'clear', 'copy', 'difference', 'difference_update', 'discard',
 'intersection', 'intersection_update', 'isdisjoint', 'issubset',
 'issuperset', 'pop', 'remove', 'symmetric_difference',
 'symmetric_difference_update', 'union', 'update']
>>>
```

Chúng ta sẽ tìm hiểu các phương thức này trong phần tiếp theo của chương.

## 2. Thêm bớt phần tử của tập hợp

### a. Bỏ sung thêm phần tử cho tập hợp: hàm add()

Hàm add() sẽ bỏ sung một phần tử vào tập hợp.

```
>>> A = set() # tạo tập hợp rỗng
>>> A.add(1) # bỏ sung số 1
>>> A.add("a") # bỏ sung chữ cái "a"
>>> A
{1, 'a'}
>>>
```

### b. Xóa, rút bớt phần tử của tập hợp: hàm remove()

Hàm remove() sẽ xóa một phần tử từ tập hợp đang làm việc.

Cú pháp của lệnh là:

```
<set>.remove(<phần tử cần xóa>).
```

Quan sát ví dụ sau. Chú ý nếu <phần tử> trong tham số của hàm remove() không có trong tập hợp thì chương trình báo lỗi.

```
>>> A = {1, 2, "One", "Two", (1, 2, 3), (1, 2)}
>>> A.remove(1) # xóa phần tử 1
>>> A.remove(10) # lỗi vì phần tử cần xóa không có.
Traceback (most recent call last):
 File "<pyshe11#63>", line 1, in <module>
 A.remove(10)
KeyError: 10
>>> A
{(1, 2), 2, 'Two', (1, 2, 3), 'One'}
>>>
```

Chú ý: Tập hợp cũng là kiểu dữ liệu container nên nó tuân thủ theo cách gán tên theo namespace giống như list hay dict.

Ví dụ sau chỉ ra điều đó. Nếu thực hiện lệnh `C = A` thì cả 2 tên A, C đều chỉ vào một đối tượng tập hợp duy nhất. Do vậy sau khi bổ sung vào A một phần tử thì C vẫn chính là A.

```
>>> C = A
>>> A
{2, 3, 7, 'One'}
>>> A.add(10)
>>> A
{2, 3, 7, 10, 'One'}
>>> C
{2, 3, 7, 10, 'One'}
>>>
```

Bây giờ nếu dùng hàm `copy()` để tạo ra một bản sao tập hợp, một đối tượng khác thì điều trên sẽ không đúng nữa.

Quan sát ví dụ sau, sau khi gán `B = A.copy()` thì A, B là các tên chỉ vào các đối tượng khác nhau. Do đó khi B thay đổi thì A vẫn như cũ.

```
>>> A = {2, 3, 7, 'One'}
>>> B = A.copy()
>>> B
{2, 3, 'One', 7}
>>> B.add(1)
>>> B
{1, 2, 3, 7, 'One'}
>>> A
{2, 3, 7, 'One'}
```

### c. Xóa tất cả, làm rỗng tập hợp

Hàm `clear()` sẽ xóa, làm rỗng tập hợp hiện thời.

```
>>> A = {'', 'N', 'a', '?', '8', '}', 'd', 'X'}
>>> A.clear()
>>> A
set()
A đã trở thành tập hợp rỗng.
>>>
```

## 3. Các phép toán trên tập hợp

Python hỗ trợ các phép toán trên tập hợp như sau:

- Phép hợp (union).
- Phép giao (intersection).
- Phép trừ (difference).
- Phép trừ đối xứng (symmetric difference).

### a. Kiểm tra quan hệ nằm trong, tập con, tập chứa tập hợp hay không

Toán tử **in** dùng để kiểm tra một phần tử có nằm trong một tập hợp hay không.

```
>>> A
{(1, 2), 2, 'Two', (1, 2, 3), 'One'}
>>> 2 in A
True
>>> 1 in A
False
>>>
```

Hàm **issubset()** kiểm tra xem một tập hợp có nằm trong tập hợp khác hay không.

**B.issubset(A)** kiểm tra B có là tập hợp con của A hay không, trả lại đúng hoặc sai.

Ngược lại hàm **issuperset()** kiểm tra xem một tập hợp có chứa (bao hàm) tập hợp khác hay không.

**A.issuperset()** kiểm tra xem tập hợp A có chứa (bao hàm) tập hợp B hay không. Hàm sẽ trả lại giá trị đúng hoặc sai.

Xem các ví dụ sau.

```
>>> A = {(1, 2), 2, 'Two', (1, 2, 3), 'One'}
>>> B = {2, "One"}
>>> B.issubset(A) # kiểm tra B có là tập hợp con của A
True
>>> A.issuperset(B) # kiểm tra A có chứa tập hợp B
True
>>>
```

### b. Phép hợp của tập hợp (|)

Phép hợp của tập hợp có thể dùng hàm **union()** hoặc ký hiệu **"|"**.

Theo định nghĩa  $A \cup B$  = tập hợp các phần tử nằm trong A hoặc B.

Chú ý: Cả 2 phương thức **union()** và phép toán **|** sẽ tạo ra một tập hợp mới, không làm thay đổi tập hợp gốc. Ví dụ:

```
>>> A = {1, 3, 5, 6, 7, 8}
>>> B = {1, 2, 8, 9, 10}
>>> A | B
{1, 2, 3, 5, 6, 7, 8, 9, 10}
>>> A.union(B)
{1, 2, 3, 5, 6, 7, 8, 9, 10}
>>>
```

Các lệnh này tạo ra tập hợp mới không làm thay đổi A và B.

### c. Phép giao tập hợp (&)

Phép giao của tập hợp có thể dùng hàm **intersection()** hoặc ký hiệu **"&"**.

Theo định nghĩa  $A \cap B$  = tập hợp các phần tử nằm đồng thời trong A và B.

Chú ý: Cả 2 phương thức **intersection()** và phép toán **&** sẽ tạo ra một tập hợp mới, không làm thay đổi tập hợp gốc. Ví dụ:



```
>>> A = {1, 3, 5, 6, 7, 8}
>>> B = {1, 2, 8, 9, 10}
>>> A & B
{8, 1}
>>> A.intersection(B)
{8, 1}
```

Các lệnh này tạo ra tập hợp mới, không làm thay đổi A và B.

#### d. Phép trừ (hiệu) tập hợp (-)

Phép trừ của tập hợp có thể dùng hàm difference() hoặc ký hiệu “-”

Theo định nghĩa  $A - B$  = tập hợp các phần tử nằm đồng thời nằm trong A và không nằm trong B. Chú ý: Cả 2 phương thức difference() và phép toán - sẽ tạo ra một tập hợp mới, không làm thay đổi tập hợp gốc. Ví dụ:

```
>>> A = {1, 3, 5, 6, 7, 8}
>>> B = {1, 2, 8, 9, 10}
>>> A - B
{3, 5, 6, 7}
>>> B - A
{9, 2, 10}
```

Các lệnh này tạo ra tập hợp mới, không làm thay đổi A và B.

#### e. Phép trừ đối xứng (^)

Phép trừ đối xứng của tập hợp có thể dùng hàm symmetric\_difference() hoặc ký hiệu “^”

Theo định nghĩa  $A \wedge B$  = tập hợp các phần tử hoặc nằm trong A nhưng không nằm trong B hoặc nằm trong B nhưng không nằm trong A.

Ta có công thức tính  $A \wedge B$  như sau:

$$A \wedge B = (A - B) \cup (B - A)$$

hoặc có thể viết cách khác.

$$A \wedge B = (A \cup B) - (A \cap B)$$

Ví dụ:

```
>>> A = {1, 3, 5, 6, 7, 8}
>>> B = {1, 2, 8, 9, 10}
>>> A ^ B
{2, 3, 5, 6, 7, 9, 10}
```

## 4. Tổng kết các kiểu dữ liệu trên Python đã học

Tới đây chúng ta sẽ có một tổng kết ngắn về các kiểu dữ liệu đã được học trong cuốn sách này. Tất cả các kiểu dữ liệu của Python được chia làm 2 loại: Có thể thay đổi được (mutable) và không thay đổi được (immutable).

Bảng các kiểu dữ liệu đó như sau:

|           |                              |
|-----------|------------------------------|
| immutable | int, float, str, bool, tuple |
| mutable   | list, dict, set              |

Chú ý:

- Các thành phần của **list** và **tuple** có thể là bất kỳ kiểu dữ liệu nào: Thay đổi được hay không thay đổi được.

- Các thành phần của **set** có thể là bất kỳ trừ ra **set**, **list** và **dict**. Vậy các phần tử của **set** phải là **immutable**<sup>9</sup>.

- Với **dict** thì khóa (key) phải là dữ liệu **immutable**, còn giá trị (value) thì không hạn chế kiểu dữ liệu.

Bảng tổng kết các lệnh và toán tử trên dữ liệu tập hợp.

| Phép toán tập hợp        | Phương thức               | Toán tử                |
|--------------------------|---------------------------|------------------------|
| A giao B                 | A.intersection(B)         | A & B<br>A & B = B & A |
| A hợp B                  | A.union(B)                | A   B<br>A   B = B   A |
| A trừ B                  | A.difference(B)           | A - B                  |
| Phép trừ đối xứng A và B | A.symmetric_difference(B) | A ^ B<br>A ^ B = B ^ A |
| Tập hợp con              | A.issubset(B)             | A <= B                 |
| Tập hợp cha              | A.issuperset(B)           | A >= B                 |



## Câu hỏi - bài tập

1. Viết chương trình tạo tập hợp A bao gồm 5 phần tử ngẫu nhiên lấy từ dãy 0, 1, 2, 3, ..., 9.

2. Đoạn chương trình sau sẽ in ra kết quả gì?

```
1. from random import *
2. A = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3. C = set()
4. for i in range(5):
5. x = choice(A)
6. C.add(x)
7. A.remove(x)
8. print(C)
```

So sánh kết quả này với bài tập 1.

3. Đoạn chương trình sau cho kết quả gì?

```
>>> import random as r
>>> C = {0, 1, 2, 3, 4, 5}
>>> x = r.choice(C)
>>> print(x)
```

<sup>9</sup> Thật ra set có thể chứa tất cả các phần tử là hashable (có thể băm được).

- a) 0
- b) 5
- c) Một giá trị ngẫu nhiên trong khoảng 0, 1, 2, 3, 4, 5.
- d) Chương trình báo lỗi.

4. Giả sử cho tập hợp C như sau:

$C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ .

Viết chương trình tạo ra tập hợp A bao gồm 5 phần tử lấy ngẫu nhiên từ tập hợp C.

5. Cho tập hợp C bao gồm các phần tử chỉ là số hoặc xâu ký tự. Viết chương trình tách tập hợp C thành 2 tập con N và S:

- N là tập con của C bao gồm các phần tử là số (nguyên hoặc thập phân).

- S là tập con của C bao gồm các phần tử là xâu ký tự.

6. Giải thích ý nghĩa của chương trình sau:

---

```
1. C = set()
2. n = int(input("Nhập số tự nhiên n: "))
3. def isPrime(p):
4. if p == 1:
5. return False
6. else:
7. for k in range(2, p):
8. if p % k == 0:
9. return False
10. return True
11. for k in range(1, n):
12. if isPrime(k):
13. C.add(k)
14. print(C)
```

---

7. Dựa trên kết quả của bài 6, viết chương trình thực hiện công việc sau:

- Nhập từ bàn phím số tự nhiên n.

- In ra màn hình dãy n số nguyên tố đầu tiên.

8. Viết chương trình sinh ngẫu nhiên 2 tập hợp số tự nhiên A, B. Sau đó liệt kê ra màn hình các phần tử chung của A, B.

9. Viết chương trình sinh ngẫu nhiên 2 tập hợp số tự nhiên A, B. Sau đó liệt kê ra màn hình các phần tử thuộc A nhưng không thuộc B.

10. Viết chương trình sinh ngẫu nhiên 2 tập hợp số tự nhiên A, B. Sau đó liệt kê ra màn hình các phần tử hoặc thuộc A nhưng không thuộc B hoặc thuộc B nhưng không thuộc A.

11. Cho đoạn chương trình sau:

---

```
>>> A = {0, 2, 3, 6, 9, 10, 11}
>>> B = {0, 1, 4, 6, 7, 8, 9, 12}
>>> C = A&B
```

---

Tập hợp C bằng bao nhiêu?

- a) {0, 1, 3, 6, 8, 9}

- b) {0, 6, 9}
- c) {0, 1, 6, 9, 10}
- d) {0}

12. Cho đoạn chương trình sau:

---

```
>>> A = {0, 2, 3, 6, 9, 10, 11}
>>> B = {0, 1, 4, 6, 7, 8, 9, 12}
>>> C = A - B
```

---

Tập hợp C bằng bao nhiêu?

- a) {0, 1, 3, 6, 8, 9, 10}
- b) {0, 10, 6, 9}
- c) {10, 11, 2, 3}
- d) {0, 1, 2, 3}

13. Cho đoạn chương trình sau:

---

```
>>> A = {0, 2, 3, 6, 9, 10, 11}
>>> B = {0, 1, 4, 6, 7, 8, 9, 12}
>>> C = A^B
```

---

Tập hợp C bằng bao nhiêu?

- a) {1, 2, 3, 4, 7, 8, 10, 11, 12}
- b) {0, 1, 2, 3, 4, 6, 7, 8, 9}
- c) {1, 2, 3, 4, 10, 11, 12, 3}
- d) {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

14. Quan sát chương trình sau:

---

```
1. n = int(input("Nhập số n: "))
2. C = set()
3. def isPrime(p):
4. if p == 1:
5. return False
6. else:
7. for k in range(2, p):
8. if p % k == 0:
9. return False
10. return True
11. k = 2
12. while k <= n:
13. if n%k == 0:
14. C.add(k)
15. while n%k == 0:
16. n = n // k
17. else:
18. k = k + 1
19. print(C)
```

---

Kết quả tập hợp C chứa các giá trị nào?

15. Viết chương trình sinh các tập hợp A, B, C như sau:

- A là tập hợp 100 số tự nhiên ngẫu nhiên trong khoảng [1, 1000].
- B là tập hợp 20 số tự nhiên ngẫu nhiên lấy từ tập hợp A.
- C là tập hợp 10 số tự nhiên ngẫu nhiên lấy từ tập hợp B.

16. Viết chương trình sinh các tập hợp A, B có ý nghĩa như sau:

- A là tập hợp 10 số tự nhiên ngẫu nhiên lấy từ khoảng [2, 1000].
- B là tập hợp tất cả các số là ước số nguyên tố của các số lấy từ A. Điều đó có nghĩa là số p thuộc B khi và chỉ khi: p là số nguyên tố và p là ước số của một số nằm trong A. Ở đây hiệu ước số của một số có thể là 1 và chính nó.

17. Cho trước tập hợp C bao gồm cả số nguyên, số thực và xâu ký tự. Viết chương trình đếm số phần tử là nguyên, thập phân và xâu ký tự của C.

18. Viết chương trình thực hiện công việc sau:

- Nhập một số tự nhiên n từ bàn phím.

- Tạo 2 tập hợp A, B: A bao gồm các số nguyên tố đồng thời là ước số của n; B bao gồm các số nguyên tố nhỏ hơn n và không là ước của n.

19. Giả sử có 2 tập hợp A, B. Đề bổ sung các phần tử thuộc B nhưng nằm ngoài A đưa vào A thì dùng lệnh nào dưới đây:

- a)  $A = A + B$
- b)  $A.union(B)$
- c)  $A | B$
- d)  $A = A.union(B)$

20. Giả sử chúng ta đã có 2 tập hợp A, B. A là tập hợp tên các bạn học sinh lớp 9A, B là họ tên các bạn học sinh lớp 9B. Ví dụ:

$A = \{"Bùi Việt Hà", "Nguyễn Quang Vinh", "Trương Anh Tú", "Trần Thị Hoa"\}$ .

$B = \{"Nguyễn Việt Hà", "Bùi Cẩm Tú", "Lê Quang Vinh", "Hà Tú Anh"\}$ .

Viết một chương trình tính:

- Số các học sinh có tên trùng nhau với lớp bên cạnh của 2 lớp. Trong ví dụ trên số học sinh trùng tên là 3 của mỗi lớp.
- Số các học sinh có họ trùng nhau với lớp bên cạnh của 2 lớp. Trong ví dụ trên số học sinh trùng họ của mỗi lớp là 2.

21. Sử dụng tính chất các hàm `set()` và `list()` có thể chuyển đổi dữ liệu từ kiểu list sang set và ngược lại, viết các hàm số sau:

- Hàm  $f(<list>)$  với đầu vào là một list, đầu ra cũng là một list nhưng đã bỏ đi các phần tử giống nhau của dãy gốc.
- Hàm  $f(<list1>, <list2>)$  với đầu vào là 2 list bất kỳ, đầu ra là một dãy bao gồm tất cả phần tử có trong 2 dãy gốc trên sau khi đã loại đi các phần tử giống nhau.



## Chương 15. Đồ họa con rùa

### Mục tiêu:

- Giới thiệu tổng quan module đồ họa con rùa.
- Một số lệnh điều khiển rùa trên màn hình và điều khiển sân khấu.



### Nội dung chính

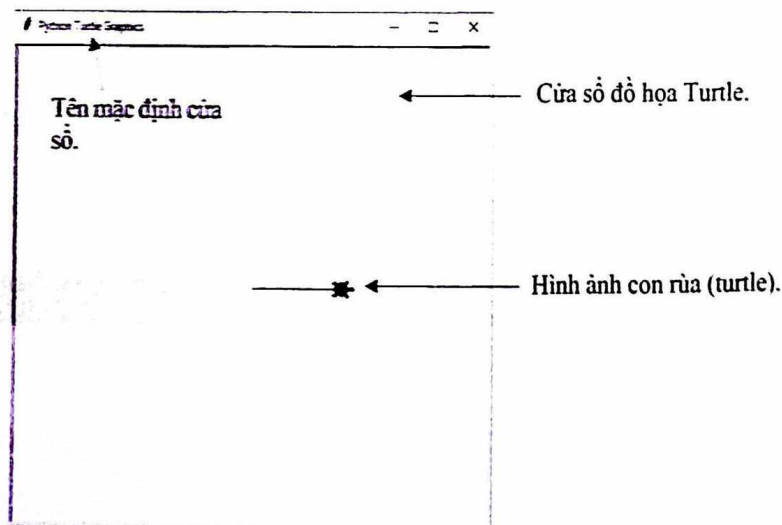
Môi trường đồ họa Turtle (con rùa) là một module hệ thống của Python. Môi trường này lấy các ý tưởng từ môi trường lập trình Logo nổi tiếng. Đối tượng hay nhân vật chính của môi trường đồ họa này là con rùa, là một thể hiện của Turtle. Khi bắt đầu khởi động môi trường đồ họa Turtle, con rùa sẽ nằm chính giữa màn hình (với tọa độ 0, 0) và hướng về bên phải (hướng  $0^\circ$ ). Hầu hết các lệnh của môi trường đồ họa này đều tập trung điều khiển nhân vật con rùa này.

### 1. Thiết lập môi trường đồ họa Turtle

Để khởi động môi trường đồ họa này chúng ta thực hiện lệnh `import turtle`. Ví dụ các lệnh sau sẽ khởi động môi trường đồ họa Turtle, cho rùa đi về phía trước khoảng cách 100 và sau đó thay đổi hình con rùa (hình mặc định của con rùa là mũi tên).

```
>>> import turtle as t
>>> t.fd(100) # đi chuyển về phía trước 100 bước.
>>> t.shape("turtle") # thay đổi thành hình con rùa
```

Chúng ta sẽ thấy xuất hiện ngay cửa sổ Python Turtle Graphics như hình dưới đây. Vị trí gốc của con Rùa nằm chính tâm màn hình. Rùa đã đi và vẽ một đoạn bằng 100 pixel về bên phải và hình ảnh con rùa xuất hiện.



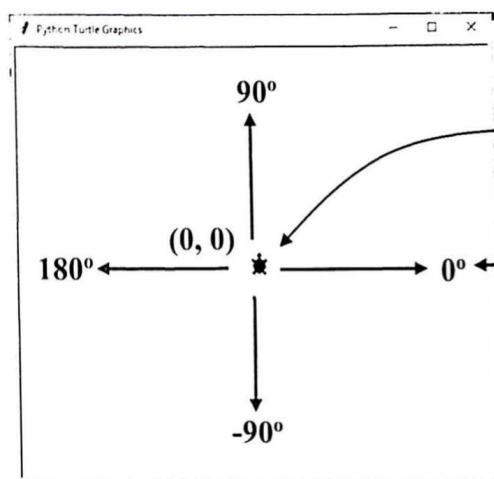
## 2. Hướng của rùa, chế độ vẽ và tọa độ màn hình

Trạng thái của rùa trên màn hình được xác định bởi 3 yếu tố:

- Hướng nhìn (heading).
- Tọa độ X.
- Tọa độ Y.

Khi di chuyển, rùa sẽ không để lại vết nếu chế độ vẽ là “nâng bút” (penup), ngược lại là chế độ “hạ bút” (pendown), khi di chuyển rùa sẽ vẽ vết trên màn hình. Chế độ mặc định của rùa là hạ bút.

Khung vẽ (canvas) là khung màn hình mà rùa có thể chuyển động bên trong. Kích thước mặc định là 640 x 480. Tâm màn hình có tọa độ (0, 0).



Tọa độ của rùa. Mặc định tọa độ = (0, 0).

Hướng (heading) của rùa. Mặc định hướng là 0 độ.

Sau đây là một số lệnh trả về các thông số của vị trí hiện thời của con rùa trên màn hình.

### Lệnh

### Ý nghĩa

heading()

Hướng hiện thời của rùa. Giá trị này tính theo độ, từ 0 đến 360 độ.

xcor()

Tọa độ x hiện thời của rùa.

ycor()

Tọa độ y hiện thời của rùa.

penup() | up | pu

Thiết lập chế độ nâng bút, không vẽ khi rùa di chuyển.

pendown() | down | pd

Thiết lập chế độ hạ bút, rùa sẽ vẽ khi di chuyển.

isdown()

Trả về trạng thái bút vẽ, là True nếu đang hạ bút và False nếu đang nâng bút.

**Chú ý:** Để thiết lập chế độ làm việc với đồ họa con rùa chúng ta có thể thực hiện theo nhiều cách.

Cách đơn giản nhất là thiết lập một thể hiện (instance) của turtle, ví dụ t, và làm việc với đối tượng rùa này.

```
>>> import turtle as t
>>> t.shape("turtle")
```

Một cách khác đơn giản hơn.

```
>>> from turtle import *
>>> shape("turtle")
```

Chúng ta sẽ thường xuyên dùng cả 2 cách trên.

### 3. Các lệnh điều khiển rùa trên màn hình

Bảng sau mô tả các lệnh điều khiển rùa chính trên màn hình. Các lệnh này hoàn toàn đồng nhất với các lệnh điều khiển rùa trong LOGO.

| Lệnh                     | Ý nghĩa                                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------------------------------|
| fd(d)   forward          | Di chuyển về phía trước khoảng cách d, d có thể là số nguyên hoặc thập phân. Nếu d < 0 thì đi lùi.          |
| back(d)   bk, backward   | Di chuyển về phía sau khoảng cách d, d có thể là số nguyên hoặc thập phân. Nếu d < 0 thì đi lên phía trước. |
| left(alpha)   lt         | Quay trái một góc với số đo alpha độ.                                                                       |
| right(alpha)   rt        | Quay phải một góc với số đo alpha độ.                                                                       |
| seth(angle)   setheading | Thiết lập hướng hiện thời của rùa theo số đo angle độ.                                                      |
| goto(x, y)   setpos      | Di chuyển nhanh đến vị trí tọa độ (x, y).                                                                   |
| setx(x)                  | Thiết lập tọa độ X của rùa = x.                                                                             |
| sety(y)                  | Thiết lập tọa độ Y của rùa = y.                                                                             |
| home()                   | Đưa rùa về tọa độ gốc và hướng về 0.                                                                        |
| reset()                  | Xóa màn hình và đưa rùa về chế độ mặc định.                                                                 |

Xét ví dụ đơn giản sau. Thiết lập thủ tục square(d) để vẽ hình vuông cạnh d.

```
>>> import turtle as t
>>> t.pendown()
>>> def square(d):
 for i in range(4):
 t.fd(d)
 t.left(90)
>>> square(100)
>>>
```



kết quả của lệnh

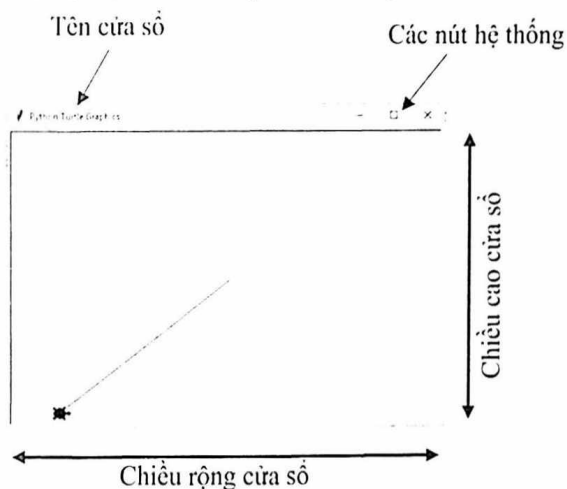
## 4. Thiết lập kích thước sân khấu và hệ tọa độ

Trước tiên chúng ta cùng xác định một số thuật ngữ quan trọng liên quan đến môi trường đồ họa Turtle của Python.

### 4.1. Phân biệt cửa sổ (window) Turtle và vùng vẽ (canvas)

#### a) Cửa sổ đồ họa Turtle

Cửa sổ đồ họa Turtle là khung cửa sổ làm việc trong chế độ đồ họa. Khung này có các thông số như tên cửa sổ, chiều rộng, chiều cao và vị trí tương đối so với màn hình máy tính. Mặc định cửa sổ này nằm chính giữa màn hình, có kích thước 640 x 480.



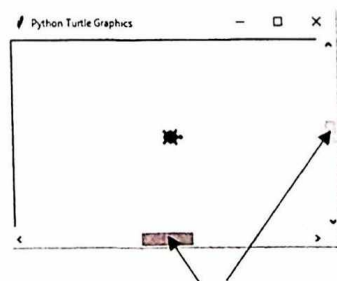
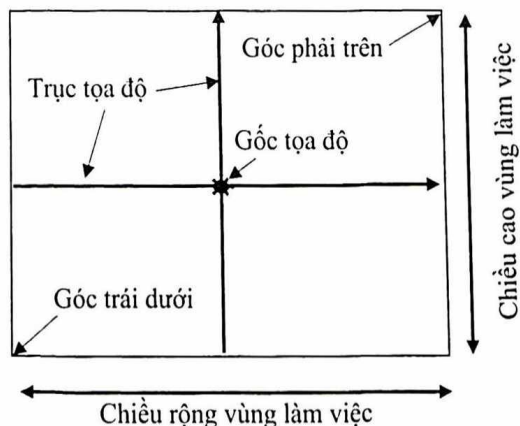
Vị trí của cửa sổ đồ họa Turtle trên màn hình máy tính.



#### b) Vùng làm việc (canvas)

Vùng làm việc (canvas) là vùng tọa độ logic mà nhân vật rùa có thể di chuyển bên trong. Vùng này được xác định bởi các thông số như chiều rộng, chiều cao, màu nền màn hình. Trong vùng làm việc này sẽ phải thiết lập một hệ trục tọa độ (X, Y). Mặc định tâm của hệ tọa độ này nằm chính giữa vùng canvas.

Chú ý: Vùng làm việc (canvas) có thể có kích thước khác với kích thước cửa sổ đồ họa đã nêu trong mục trên.



Nếu vùng làm việc lớn hơn kích thước cửa sổ thì sẽ xuất hiện các thanh cuộn.

## 4.2. Lệnh `setup()`

Lệnh này sẽ thiết lập các thông số cửa sổ đồ họa Turtle. Các thông số bao gồm:

- Chiều rộng.
- Chiều cao.
- Vị trí tương đối so với màn hình máy tính.

Cú pháp lệnh như sau.

---

```
setup(width=0.5, height=0.75, startx=None, starty=None)
```

---

Ý nghĩa của lệnh mô tả trong bảng sau.

| Lệnh                                                     | Mô tả ý nghĩa                                                                                                                                                                      |
|----------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setup()</code>                                     | Thiết lập chế độ và thông số mặc định của cửa sổ đồ họa: kích thước 640 x 480 và nằm giữa màn hình.                                                                                |
| <code>setup(400, 300)</code>                             | Thiết lập kích thước cửa sổ là 400 x 300.                                                                                                                                          |
| <code>setup(0.5, 0.75)</code>                            | Thiết lập kích thước cửa sổ đồ họa: <ul style="list-style-type: none"><li>- Chiều rộng = 0.5 của chiều rộng màn hình.</li><li>- Chiều cao = 0.75 của chiều cao màn hình.</li></ul> |
| <code>setup(480, 360, startx = 100, starty = 200)</code> | Thiết lập cửa sổ đồ họa kích thước 480 x 360 và nằm ở vị trí cách bên trái màn hình 100 và cách phía trên 200.                                                                     |

## 4.3. Lệnh `screensize()`

Lệnh này sẽ thiết lập các thông số cho vùng làm việc (canvas) hiện thời. Các thông số của vùng làm việc bao gồm:

- Chiều rộng (canvwidth), tính theo pixel.
- Chiều cao (canvheight), tính theo pixel.

Cú pháp của lệnh:

---

```
screensize(canvwidth=None, canvheight=None, bg=None)
```

---

Chú ý: Lệnh này không tác động lên các thông số và vị trí của cửa sổ đồ họa Turtle.

Ý nghĩa của lệnh mô tả trong bảng sau.

| Lệnh                                 | Mô tả ý nghĩa                                                 |
|--------------------------------------|---------------------------------------------------------------|
| <code>screensize()</code>            | Trả về thông tin kích thước vùng làm việc (canvas) hiện thời. |
| <code>screensize(bg = "blue")</code> | Thiết lập màu nền "xanh" cho vùng làm việc hiện thời.         |



| Lệnh                                | Mô tả ý nghĩa                                                          |
|-------------------------------------|------------------------------------------------------------------------|
| <code>screenSize(400, 300)</code>   | Thiết lập kích thước 400 x 300 cho vùng làm việc (canvas) hiện thời.   |
| <code>screenSize(2000, 1500)</code> | Thiết lập kích thước 2000 x 1500 cho vùng làm việc (canvas) hiện thời. |

#### 4.4. Lệnh `setworldcoordinates()`

Lệnh này sẽ thiết lập một hệ tọa độ ảo cho vùng làm việc hiện thời. Hệ tọa độ này sẽ được áp đặt lên vùng canvas hiện thời. Nếu áp dụng lệnh này thì toàn bộ hình ảnh trên màn hình sẽ được vẽ lại trong hệ tọa độ mới.

Cú pháp của lệnh như sau.

---

```
setworldcoordinates(llx, lly, urx, ury)
```

---

Trong đó:

`llx` = tọa độ x của điểm góc trái dưới vùng làm việc.

`lly` = tọa độ y của điểm góc trái dưới vùng làm việc.

`urx` = tọa độ x của điểm góc phải trên vùng làm việc.

`ury` = tọa độ y của điểm góc phải trên vùng làm việc.

Ý nghĩa của lệnh mô tả trong bảng sau.

| Lệnh                                                | Mô tả ý nghĩa                                                                                                                                                                                                             |
|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setworldcoordinates(-240,-180,240,180)</code> | Thiết lập hệ tọa độ với góc trái dưới là (-240, -180) và góc phải trên là (240,180). Do vậy trong hệ tọa độ này, tâm (gốc tọa độ) nằm chính giữa cửa sổ và các trục tọa độ theo chiều rộng là 480, theo chiều cao là 360. |
| <code>setworldcoordinates(0,0,300,200)</code>       | Thiết lập hệ tọa độ với góc trái dưới là (0, 0) và góc phải trên là (300, 200). Do vậy trong hệ tọa độ này, tâm nằm tại góc trái dưới và các trục tọa độ theo chiều rộng 300, theo chiều cao là 200.                      |

Một số ví dụ.

---

|                                                                                                                                             |                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&gt;&gt;&gt; import turtle as t &gt;&gt;&gt; t.setup(640, 480) &gt;&gt;&gt; t.screenSize(640,480) &gt;&gt;&gt; t.shape("turtle")</pre> | Thiết lập cửa sổ đồ họa và vùng làm việc có kích thước 640 x 480, gốc tọa độ nằm chính giữa cửa sổ. Hiện thị nhân vật là hình con rùa. |
|---------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|

---

## 5. Thiết lập bút vẽ và tô màu các hình khối

### 5.1. Vẽ hình vuông đầu tiên

Chúng ta sẽ thiết lập một hàm số đầu tiên `square()` dùng để vẽ hình vuông với độ dài cạnh cho trước. Mặc định đối tượng Turtle sẽ được ký hiệu là `t`.

Trong tất cả các minh họa của phần này chế độ đồ họa Turtle sẽ luôn được khởi động mặc định với tên gọi `t`.

Cú pháp của hàm: `square(d)` với `d` là số tự nhiên dùng để vẽ hình vuông cạnh `d`.

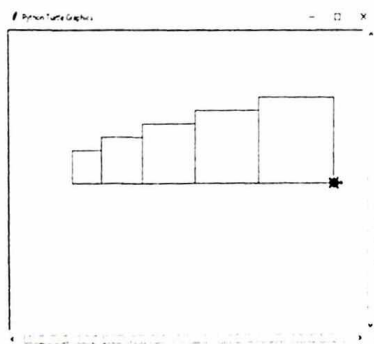
```
1. def square(d):
2. t.pendown() # hạ bút bắt đầu chế độ vẽ
3. count = 0
4. while count < 4:
5. t.fd(d)
6. t.left(90)
7. count = count + 1
8. t.penup() # nhấc bút, kết thúc chế độ vẽ.
```

Lặp 4 lần lệnh di chuyển `d` bước, sau đó quay trái góc 90 độ.

Chương trình đơn giản sau sẽ vẽ 5 hình vuông kế nhau, hình vuông sau có cạnh lớn hơn hình vuông trước 20 đơn vị. Hình vuông đầu tiên xuất phát từ tọa độ `(-200, 0)`.

#### Vẽ\_hình\_vuong.py

```
9. import turtle as t
10. t.setup(640, 480) # thiết lập kích thước của sổ
11. t.screensize(640, 480) # thiết lập hệ tọa độ theo của sổ
12. t.st() # showturtle()
13. t.shape("turtle")
14. t.penup() # chế độ nâng bút - mặc định
15. t.seth(0) # thiết lập hướng mặc định = 0 độ.
16. def square(d):
17. t.pendown()
18. count = 0
19. while count < 4:
20. t.fd(d)
21. t.left(90)
22. count = count + 1
23. t.penup()
24. def main():
25. t.goto(-200, 0)
26. d = 50
27. count = 0
28. while count < 5:
29. square(d)
30. t.fd(d)
31. d = d + 20
32. count = count + 1
33. main()
34. t.mainloop()
```



Kết quả thực hiện chương trình.

Ý nghĩa của lệnh `mainloop()` sẽ được nói sau.

## 5.2. Thiết lập màu bút vẽ

Lệnh `color()` sẽ thiết lập màu vẽ cho bút vẽ.

Cú pháp của lệnh rất đơn giản:

---

```
color(<color string>)
```

---

Ví dụ:

---

```
t.color("red")
t.color("blue")
```

---

Các string màu sắc có sẵn trong Python sẽ là:

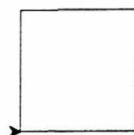
**white, black, red, cyan, blue, green, yellow, purple, magenta, violet.**

Ví dụ một chương trình đơn giản.

---

```
>>> t.reset()
>>> t.color("blue")
>>> def square(d):
 t.pendown()
 for i in range(4):
 t.fd(d)
 t.left(90)
 t.penup()
>>> square(100)
```

---



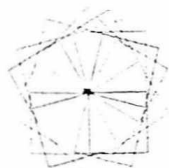
Kết quả thu được.

Chúng ta cùng làm một ví dụ phức tạp hơn. Các hình vuông sẽ được vẽ cùng từ 1 điểm duy nhất, nhưng với màu sắc thay đổi và hướng ban đầu thay đổi theo các góc là bội của 15, từ 0 đến 360 độ. Hình ảnh thu được sẽ rất đẹp mắt.

---

```
1. t.reset()
2. Colors = ["red", "blue", "cyan", "magenta", "violet"]
3. i = 0
4. color = Colors[i]
5. for alpha in range(0, 360, 15):
6. t.color(color)
7. t.seth(alpha)
8. square(100)
9. i = (i + 1) % len(Colors)
10. color = Colors[i]
```

---



Kết quả thu được.

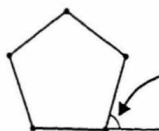
## 5.3. Vẽ đa giác đều

Hàm vẽ đa giác đều `polygon()` sẽ có 2 tham số:  $d$  = độ dài cạnh và  $n$  = số đỉnh của đa giác.

---

```
1. def polygon(d, n):
2. t.pendown()
3. for i in range(n):
4. t.fd(d)
5. t.left(360/n)
6. t.penup()
```

---



Góc này  
 $= 360^\circ/n$

Đoạn chương trình sau sẽ vẽ 5 hình ngũ giác đều lồng nhau.

```
>>> t.reset()
>>> d = 50
>>> for i in range(5):
 polygon(d, 5)
 d = d + 20
```



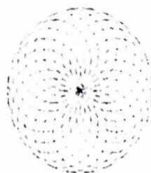
#### 5.4. Hàm circle()

Hàm circle() được xây dựng sẵn trong Turtle module dùng để vẽ hình tròn, cung tròn và các hình đa giác đều.

Với cú pháp đơn giản nhất, circle(d) sẽ vẽ hình tròn bán kính d từ vị trí hiện thời. Nếu  $d > 0$ , hình tròn được vẽ ngược kim đồng hồ, nếu  $d < 0$ , hình tròn được vẽ có bán kính  $|d|$  xuôi kim đồng hồ.

Xét ví dụ sau vẽ chùm các hình tròn bán kính 100 và xoay xung quanh gốc tọa độ, góc quay = 15 độ.

```
>>> t.reset()
>>> for alpha in range(0, 360, 15):
 t.seth(alpha)
 t.circle(100)
```



Muốn vẽ một cung tròn, chúng ta bổ sung thêm một tham số góc để xác định giới hạn cung tròn đó.

circle(d, <góc>) sẽ vẽ cung tròn, xuất phát từ vị trí hiện thời, bán kính d và có góc cung bằng <góc>. Ví dụ:

circle(d, 180) sẽ vẽ nửa vòng tròn.

circle(d, 90) sẽ vẽ một phần tư vòng tròn.

Ví dụ một chương trình tương tự trên nhưng vẽ nửa vòng tròn.

```
>>> t.clear()
>>> for alpha in range(0, 360, 15):
 t.seth(alpha)
 t.penup()
 t.goto(0, 0)
 t.pendown()
 t.circle(100, 180)
```



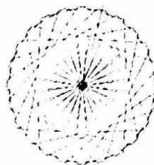
Hàm circle() cũng có thể dùng để vẽ đa giác đều. Cú pháp vẽ đa giác đều như sau:

```
circle(d, steps = <n>)
```

Lệnh này sẽ vẽ đa giác đều n cạnh với bán kính vòng tròn ngoại tiếp d.

Ví dụ phỏng theo cách vẽ trên ta có chương trình sau.

```
>>> t.clear()
>>> for alpha in range(0,360,15):
 t.seth(alpha)
 t.goto(0,0)
 t.circle(100, steps = 5)
```



### 5.5. Tô màu hình được vẽ

Để tô màu được hình vẽ (hình có thể là đóng kín hoặc hở) chúng ta cần thiết lập màu nền để tô màu. Màu bút và màu nền được thiết lập bởi lệnh `color()`. Cú pháp đầy đủ của lệnh `color()` thiết lập màu sắc như sau:

```
color(<màu bút>, <màu nền>)
```

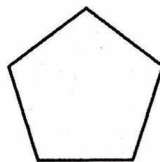
Nếu tham số <màu nền> vắng mặt thì Python hiểu màu nền và màu bút giống nhau.

Lệnh `color()` không có tham số sẽ trả lại màu bút và màu nền hiện thời.

Muốn tô màu một hình đang vẽ, chúng ta chèn các lệnh vẽ này vào giữa 2 lệnh `begin_fill()` và `end_fill()`.

Chương trình đơn giản sau thiết lập màu bút là đen (black), màu nền là tím (violet), vẽ một ngũ giác đều có tô màu.

```
>>> t.reset()
>>> t.color("black","violet")
>>> t.color()
('black', 'violet')
>>> t.pensize(3)
>>> t.begin_fill()
>>> polygon(100,5)
>>> t.end_fill()
```



## 6. Thiết lập chế độ tương tác đơn giản trên màn hình

Trong phần này chúng ta sẽ làm quen với các lệnh để có thể thiết lập được một ứng dụng đồ họa tuy đơn giản nhưng hoàn chỉnh.

Một trong những tính chất cơ bản của một ứng dụng đồ họa là khả năng thực hiện các lệnh tương ứng với các sự kiện xảy ra trên màn hình đồ họa. Một trong những sự kiện thường gặp nhất trên màn hình là nhấp chuột (mouse click).

Như vậy muốn thiết lập một ứng dụng đồ họa dù là đơn giản nhất sẽ cần các yếu tố sau:

- (1) Thiết lập giao diện ứng dụng của đồ họa. Giao diện này thông thường là kích thước màn hình và tên của ứng dụng.
- (2) Cần một cơ chế để thực hiện các hàm số trả lời các sự kiện trên màn hình, ví dụ sự kiện người dùng nhấp chuột.
- (3) Cần cơ chế để kết thúc ứng dụng. Ví dụ khi người dùng nhấp lên một nút lệnh để đóng cửa sổ đồ họa, kết thúc chương trình.

Công việc (1) chúng ta đã biết trong các phần trước của bài học. Với công việc (3), người dùng sẽ nhấp lên nút **Exit** ở góc phải trên của màn hình để đóng ứng dụng. Chúng ta sẽ tìm hiểu các lệnh và cơ chế để thực hiện công việc (2).



## 6.1. Tham số của hàm là hàm

Trong phần học về kiểu dữ liệu danh sách (list) chúng ta đã biết Python cho phép có nhiều tên (tên biến nhớ) cùng trỏ vào một vùng bộ nhớ chính của list này. Ví dụ, giả sử A được định nghĩa là một list, khi gán B = A thì cả 2 tên A và B đều được chỉ vào vùng nhớ của list này.

Với hàm số cũng vậy Python cho phép có nhiều tên cùng chỉ vào một hàm số. Ví dụ xét hàm print(). Khi viết lệnh, ví dụ:

```
>>> print("Xin chào")
```

thì đó là lời gọi hàm. Trong khi đó viết **print** không có dấu ( ) sẽ là một tên trỏ đến hàm số đó. Chúng ta cùng kiểm tra điều này.

```
>>> print
<built-in function print>
>>>
```

Bây giờ chúng ta có thể gán tên **print** bằng một tên khác, ví dụ **ham\_in = print**, khi đó cả 2 tên **print** và **ham\_in** đều chỉ đến cùng một hàm số và chúng ta có thể thực hiện "hàm" mới này như hàm cũ print().

```
>>> ham_in = print
>>> ham_in
<built-in function print>
>>> ham_in("Xin chào")
Xin chào
>>>
```

Chúng ta có thể thực hiện **ham\_in()** như một hàm số bình thường và có chức năng như hàm **print()** vì 2 tên này, **print** và **ham\_in** cùng chỉ vào một hàm số.

Như vậy với tính chất trên, chúng ta có thể thực hiện thiết lập tham số của hàm là một hàm số khác. Python có một cơ chế gọi hàm đặc biệt: Gọi hàm thông qua một hàm số khác. Cơ chế gọi hàm như vậy chính là việc cho phép đặt hàm như một tham số của một hàm số khác.

Ví dụ định nghĩa một hàm số, trong đó tham biết là lời gọi hàm đến hàm số khác.

```
>>> def funcall(func, arg):
 return func(arg)
```

Khi gọi hàm **funcall()**, chương trình sẽ chuyển lời gọi đến hàm số là tên của tham số, hàm **func()**.

Với ví dụ hàm trên, lời gọi hàm thực tế có thể như sau.

```
>>> funcall(print, "Xin chào")
Xin chào
>>> funcall(ham_in, "Hello World")
Hello World
>>>
```

Như vậy khi gọi **funcall(print, "Xin chào")**, hàm này có 2 tham số, trong đó tham số đầu tiên chính là tên được trỏ đến hàm **print()**.

## 6.2. Hàm sự kiện của chế độ đồ họa Python

Hàm xử lý sự kiện đầu tiên sẽ được học là hàm **onscreenclick()** dùng để nhận biết sự kiện người dùng nhấp chuột lên màn hình.

Cú pháp của lệnh sự kiện này như sau:

---

```
onscreenclick(func)
```

---

Trong đó **func** là tên của hàm số phải có 2 tham số dạng (x, y).

Khi người dùng nhấp chuột trên màn hình, hàm sự kiện trên sẽ được gọi và truyền 2 tham số là tọa độ (x, y) của vị trí con trỏ chuột cho hàm số tương ứng với tên **func**.

Ví dụ chúng ta định nghĩa hàm **printxy(x, y)** như sau:

---

```
def printxy(x, y):
 t.goto(x, y)
 print("Đã click chuột:", x, y)
```

---

Ý nghĩa của hàm này như sau: Di chuyển con rùa đến vị trí (x, y) và viết ra trên cửa sổ tương tác của Python dòng chữ "Đã click chuột tại: x, y".

Bây giờ chương trình sẽ được hoàn thiện khi chúng ta chèn 2 lệnh sau đây.

---

```
t.onscreenclick(printxy)
t.mainloop()
```

---

- Lệnh **onscreenclick(printxy)** sẽ tạo "bẫy" nhận biết sự kiện nhấp chuột của người dùng. Khi có sự kiện, hàm **onscreenclick(printxy)** sẽ truyền 2 giá trị (x, y) cho hàm đối số **printxy()**.

- Lệnh **mainloop()** sẽ tạo một vòng lặp vô hạn (loop vô hạn) để luôn chờ bắt bất kỳ sự kiện nhấp chuột. Lệnh này luôn phải nằm cuối chương trình.

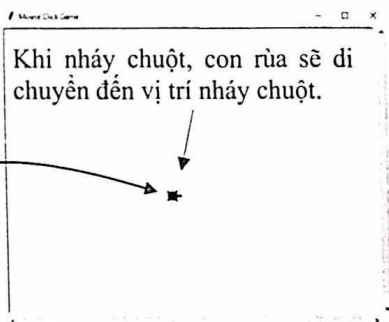
Chúng ta cùng làm một chương trình hoàn chỉnh đầu tiên.

### ClickChuot-1.py

---

```
1. import turtle as t
2. t.setup(640, 480)
3. t.screensize(640, 480)
4. t.title("Mouse Click Game")
5. t.st()
6. t.shape("turtle")
7. t.penup()
8. t.seth(0)
9. def printxy(x, y):
10. t.goto(x, y)
11. print("Đã click chuột:", x, y)
12. t.onscreenclick(printxy)
13. t.mainloop()
```

---



Lệnh **mainloop()** bắt buộc phải để ở cuối chương trình.

Debug I/O   Python Shell   Messages   OS Commands

Debug I/O (stdin, stdout, stderr) appears below

Đã click chuột: -205.0 145.0

Đã click chuột: -151.0 33.0

Đã click chuột: 20.0 64.0

Đã click chuột: -151.0 14.0

Chú ý: Khi người dùng nhấp chuột, dòng chữ "Đã click chuột: x, y" sẽ hiện trên cửa sổ tương tác của Python.

Bây giờ chúng ta sẽ xét một hàm sự kiện nữa.

```
onclick(func)
```

Hàm này sẽ bắt sự kiện khi người dùng nhấp chuột lên con rùa. Khi nhận sự kiện, hàm này sẽ truyền 2 tham số (x, y) cho hàm **func** nằm trong tham số của hàm. Cách làm việc tương tự như đã thực hiện ở trên. Chú ý rằng sự kiện **onclick()** là độc lập với sự kiện **onscreenclick()**.

Chúng ta sẽ thiết kế một ứng dụng đồ họa hoàn chỉnh nữa. Chương trình này như sau:

- Mỗi khi nhấp chuột lên màn hình, con rùa sẽ di chuyển đến vị trí vừa nhấp chuột, đồng thời trên cửa sổ Python Shell sẽ có thông báo "đã click chuột x, y".

- Mỗi khi nhấp chuột lên con rùa, rùa sẽ xoay trái 90 độ.

Cần viết 2 hàm số để chạy khi bắt được sự kiện nhấp chuột. Hàm **printxy()** dùng cho sự kiện **onscreenclick()** và hàm **turn()** dùng cho sự kiện **onclick()**.

### ClickChuot-2.py

```
1. import turtle as t
2. t.setup(640, 480)
3. t.screensize(640, 480)
4. t.title("Mouse Click Game")
5. t.st()
6. t.shape("turtle")
7. t.penup()
8. t.seth(0)
9. def printxy(x,y):
10. t.goto(x,y)
11. print("Đã click chuột:",x,y)
12. def turn(x,y):
13. t.goto(x,y)
14. t.left(90)
15. t.onclick(turn)
16. t.onscreenclick(printxy)
17. t.mainloop()
```

Các hàm số dùng để đón sự kiện.

Khi nhấp chuột lên con rùa thì rùa sẽ quay sang trái 90 độ.

Khi nhấp chuột lên màn hình thì con rùa sẽ chuyển động nhanh đến vị trí con trỏ chuột.

Bây giờ chúng ta sẽ thiết kế chương trình đồ họa đầy đủ để minh họa cho phần đồ họa con rùa.

Chương trình sẽ có các tính năng sau.

- Cửa sổ đồ họa được chia làm 3 phần theo chiều thẳng đứng. Cần kẻ các đường thẳng phân chia các vùng này.

- Mỗi khi người dùng nhấp chuột lên màn hình, chương trình sẽ vẽ một hình đặc nhỏ có tô màu.

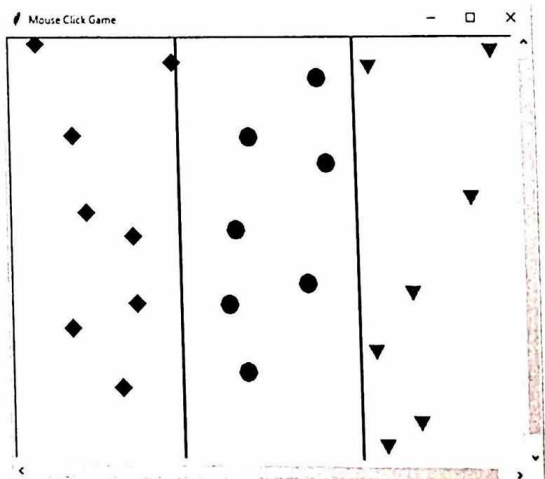
- Quy định về các hình đặc như sau: khi nhấp lên vùng trái sẽ vẽ hình vuông đỏ, khi nhấp lên vùng giữa sẽ vẽ hình tròn màu xanh, khi nhấp lên vùng bên phải thì vẽ hình tam giác màu vàng.

## GameMouse.py

```
1. import turtle as t
2. t.setup(600, 480)
3. t.screensize(600, 480)
4. t.title("Mouse Click Game")
5. t.ht()
6. t.penup()
7. t.seth(0)
8. def lines():
9. t.pensize(3)
10. for i in range(2):
11. t.goto(-100 + i*200, -240)
12. t.pendown()
13. t.goto(-100 + i*200, 240)
14. t.penup()
15. t.pensize(1)
16. Colors = ["red", "blue", "magenta"]
17. def clickdraw(x, y):
18. t.goto(x, y)
19. t.pendown()
20. t.begin_fill()
21. if x < -100:
22. t.color(Colors[0])
23. t.circle(10, steps = 4)
24. if -100 < x < 100:
25. t.color(Colors[1])
26. t.circle(10)
27. if x > 100:
28. t.color(Colors[2])
29. t.circle(10, steps = 3)
30. t.end_fill()
31. t.penup()
32. # Chương trình chính.
33. lines()
34. t.onscreenclick(clickdraw)
35. t.mainloop()
```

Giao diện của **GameMouse** như hình bên.

Khi nháy chuột lên màn hình, tùy thuộc vào vị trí trên màn hình sẽ vẽ các hình đặc khác nhau.



## Chú ý:

Muốn kết thúc chế độ chặn sự kiện, chúng ta dùng các lệnh sau:

`onclick (None)`

hoặc:

`onclick (None)`

## 6.3. Điều khiển tốc độ hoạt hình của đồ họa con rùa

### a. Điều khiển tốc độ chuyển động của rùa

Hàm `speed()` dùng để điều khiển tốc độ chuyển động của rùa (ví dụ thông qua lệnh `goto()`).

Chú pháp của hàm như sau:

`speed(<tốc độ>)`

Ở đây <tốc độ> là số nguyên bất kỳ nhưng chỉ có ý nghĩa trong khoảng từ 0 đến 10. Các tốc độ > 10 sẽ mặc định = 0.

Tốc độ mặc định của rùa là 3 (tương đối chậm).

Ý nghĩa của tốc độ như sau.

<tốc độ> =

1, rất chậm.

3, chậm.

5, trung bình.

10, nhanh.

> 10, 0, rất nhanh.

### b. Điều khiển tính năng hoạt hình

Ở chế độ vẽ, khi rùa chuyển động sẽ tạo ra vết và vẽ thành các đường đi trên màn hình đồ họa. Tính chất này gọi là tính năng hoạt hình (animation) của đồ họa con rùa, là một trong những tính năng nổi bật nhất của chương trình LOGO xưa và bây giờ của Python. Tính năng này hiện nay được áp dụng cho rất nhiều ngôn ngữ lập trình trực quan khác.

Cơ chế vẽ này như thế nào, chúng ta cùng tìm hiểu ngay bây giờ. Cơ chế hoạt hình này được xác định bởi hàm `tracer(n, delay)` với 2 tham số sau:

1. <delay>. Đây là thời gian tính theo 1/1000 giây (ms - milliseconds), sau mỗi <delay> thời gian, màn hình sẽ được tính toán lại để cập nhật hình vẽ trên cửa sổ đồ họa.

2. <n> với n là số tự nhiên > 0 hoặc = 0. Giá trị này có ý nghĩa như sau:

- Nếu  $n > 0$ . Màn hình sẽ được vẽ lại sau mỗi n lần cập nhật. Hay nói cách khác, màn hình đồ họa sẽ được vẽ lại sau mỗi  $n \times \text{<delay>}$  thời gian. Chú ý rằng thông tin màn hình được cập nhật liên tục sau mỗi <delay> thời gian, nhưng có thể chưa được thể hiện trên màn hình máy tính.

- Nếu  $n = 0$ . Màn hình sẽ không được vẽ lại một cách tự động để tạo hoạt hình. Hay nói cách khác nếu  $n = 0$  thì chúng ta sẽ hủy chế độ hoạt hình của đồ họa con rùa. Khi đó màn hình sẽ phải được vẽ lại sau khi thực hiện lệnh `update()`.



Cú pháp lệnh `tracer()` như sau.

`tracer (<n>, <delay>)`

Các chế độ đặc biệt của lệnh này.

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>tracer (0, 0)</code>  | Đặt chế độ không có hoạt hình. Tất cả các hình vẽ sẽ được thể hiện tức thời sau lệnh <code>update()</code> . |
| <code>tracer (1, 10)</code> | Chế độ hoạt hình mặc định: Màn hình sẽ được vẽ lại sau mỗi 1/100 giây (10 ms).                               |

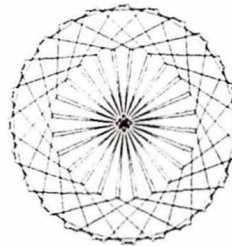
Như vậy lệnh `tracer()` chính là lệnh dùng để điều khiển hoạt hình trên màn hình. Bằng cách thay đổi các giá trị `n`, `delay` chúng ta sẽ có các kiểu hoạt hình nhanh chậm khác nhau. Để bỏ hẳn đi chế độ hoạt hình, ví dụ khi chúng ta cần vẽ nhanh một hình mà không cần xem hoạt hình, thì có thể dùng lệnh như sau.

`t.tracer(0,0)` → Đặt chế độ hủy hoạt hình.  
.....  
<các lệnh cần vẽ nhanh>  
.....  
`t.update()` → Lệnh này sẽ cập nhật màn hình sau khi vẽ xong, tốc độ vẽ sẽ gần như tức thời.

Xét ví dụ vẽ hình nhanh đơn giản sau.

#### Ve-nhanh-1.py

```
1. import turtle as t
2. t.ht()
3. t.tracer(0,0)
4. for alpha in range(0,360,15):
5. t.seth(alpha)
6. t.goto(0,0)
7. t.circle(100, steps = 5)
8. t.update()
9. t.mainloop()
```



→ Lệnh `mainloop()` để ở cuối chương trình có tác dụng không làm đóng cửa sổ đồ họa.

## 7. Tìm hiểu sâu hơn về điều khiển rùa

### 7.1. Thay đổi hình ảnh thể hiện rùa

Chúng ta đã biết một lệnh tạo/thay đổi hình ảnh nhân vật rùa trên màn hình, đó là lệnh `shape()`, ví dụ:

```
import turtle as t
t.shape("turtle")
```



Cú pháp của lệnh:

```
shape(<hình ảnh rùa>)
```

Các tên hình ảnh rùa có sẵn trong Python bao gồm:

|            |                |                                                                                   |           |               |                                                                                   |
|------------|----------------|-----------------------------------------------------------------------------------|-----------|---------------|-----------------------------------------------------------------------------------|
| "turtle"   | Hình con rùa.  |  | "circle"  | Hình tròn.    |  |
| "arrow"    | Hình mũi tên.  |  | "square"  | Hình vuông.   |  |
| "triangle" | Hình tam giác. |  | "classic" | Hình mũi tên. |  |

Ngoài các hình ảnh rùa mặc định có sẵn trên đây, chúng ta có thể tạo ra các hình ảnh khác. Muốn lấy một hình ảnh làm mẫu thể hiện của rùa chúng ta cần có tệp ảnh dạng \*.gif để lưu hình ảnh con rùa này.

Các hình ảnh rùa được lưu trong một shapelist. Lệnh **getshapes()** sẽ trả lại danh sách các tên hình ảnh rùa này. Mặc định Python hỗ trợ 7 hình ảnh như bảng trên.

```
>>> t.getshapes()
['arrow', 'blank', 'circle', 'classic', 'square', 'triangle', 'turtle']
```

Muốn đưa hình ảnh rùa từ một tệp \*.gif, trước tiên chúng ta dùng lệnh **addshape()** hoặc lệnh **register\_shape()** để đưa tên hình ảnh này vào danh sách, sau đó dùng lệnh **shape()** để hiển thị.

Ví dụ:

```
1. >>> os.chdir("D:\Habv\docx\Habv\Habv_hoc_Python\Sach-hoc-
 Python\Python-Coban\Projects\B15")
2. >>> t.addshape("Rua-1.gif")
3. >>> t.getshapes()
4. ['Rua-1.gif', 'arrow', 'blank', 'circle', 'classic', 'square',
 'triangle', 'turtle']
5. >>> t.shape("Rua-1.gif")
6. >>>
```



Lệnh 1 thiết lập đường dẫn chứa tệp Rua-1.gif. Lệnh 2 bổ sung tên "Rua-1.gif" vào danh sách các tên hình ảnh của rùa. Lệnh 5 sẽ thay đổi hình ảnh rùa sang hình ảnh của tệp gif đã có trong danh sách.

**Chú ý:**

Rùa với hình ảnh lấy từ các tệp dạng gif sẽ không thể xoay hay thay đổi kích thước trên màn hình.

## 7.2. Thay đổi kích thước và màu của rùa

Lệnh `fillcolor()` sẽ thiết lập màu của rùa.

Cú pháp:

```
fillcolor(<color>)
```

Ví dụ các lệnh sau sẽ thiết lập rùa hình con rùa với màu đỏ.

```
import turtle as t
t.shape("turtle")
t.fillcolor("red")
```



Lệnh thiết lập màu cho rùa chỉ có tác dụng với các hình ảnh là hình mẫu có sẵn trong Python hoặc hình đa diện cho người dùng thiết lập, mà không có tác dụng với hình ảnh rùa lấy từ tệp dạng gif.

Để thay đổi kích thước của rùa chúng ta dùng lệnh `shapessize()` hoặc `turtlesize()`. Cú pháp lệnh như sau.

```
shapessize(stretch_wid=None, stretch_len=None, outline=None)
```

Các tham số như sau:

`stretch_wid` = tỷ lệ co giãn theo chiều rộng (là chiều vuông góc với hướng của rùa).

`stretch_len` = tỷ lệ co giãn theo chiều ngang (là chiều cùng với hướng của rùa).

`outline` = độ rộng của viền con rùa.

Ví dụ một số lệnh thiết lập kích thước rùa.

|                                        |                                                                        |
|----------------------------------------|------------------------------------------------------------------------|
| <code>t.shapessize(2, 3)</code>        | Thiết lập chiều rộng gấp 2 và chiều dài gấp 3 lần hiện tại.            |
| <code>t.shapessize(0.5, 2, 3)</code>   | Thiết lập chiều rộng nhỏ 1/2, chiều dài gấp 2 và đường viền = 2 pixel. |
| <code>t.shapessize(outline = 4)</code> | Thiết lập đường viền = 4 pixel.                                        |

## 7.3. Phân thân của rùa

Từ một con rùa gốc, chúng ta có thể tạo ra các phân thân của rùa này để tạo nhiều rùa trên màn hình. Lệnh `clone()` có chức năng như vậy.

Xét ví dụ sau:

```
import turtle as t
t.shape("turtle")
t1 = t.clone()
```

Lệnh này sẽ tạo ra một phân thân có tên `t1` bên cạnh rùa gốc là `t`.

Bây giờ chúng ta cùng thực hiện chương trình mô tả bài toán sau.

Trên màn hình thiết lập 2 con rùa, ví dụ, ký hiệu `t` và `t1`. Rùa `t` có màu xanh, `t1` có màu đỏ. Điều khiển như sau:

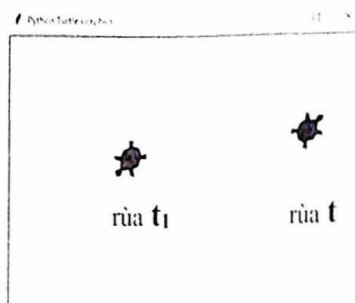
- Nháy chuột lên rùa xanh sẽ làm cho rùa này xoay 1 góc ngẫu nhiên và bước đi 50 bước.

- Nháy chuột lên màn hình sẽ điều khiển rùa đồ xoay về hướng con trỏ chuột và bước đi 20 pixel.

Chương trình điều khiển như sau.

### Clone.py

```
1. import random as r
2. import turtle as t
3. import math as m
4. t.penup()
5. t.shape("turtle")
6. t.fillcolor("blue")
7. t.shapesize(2)
8. t.goto(100,0)
9. t1 = t.clone()
10. t1.penup()
11. t1.shape("turtle")
12. t1.fillcolor("red")
13. t1.shapesize(2)
14. t1.goto(-100,0)
15. def angle(x,y): # Hàm trả lại hướng tính từ (0,0) đến (x, y)
16. if x > 0:
17. rad = m.atan(y/x)
18. alpha = rad*180/m.pi
19. elif x < 0:
20. rad = m.atan(y/x)
21. alpha = rad*180/m.pi + 180
22. else:
23. if y >= 0:
24. alpha = 90
25. else:
26. alpha = -90
27. return alpha
28. def doxy(x,y):
29. x1 = t1.xcor()
30. y1 = t1.ycor()
31. t1.seth(angle(x-x1,y-y1))
32. t1.fd(20)
33. def quickgo(x,y):
34. t.seth(r.random()*360)
35. t.fd(50)
36. # chương trình chính
37. t.onclick(quickgo)
38. t.onscreenclick(doxy)
39. t.mainloop()
```



### Chú ý quan trọng:

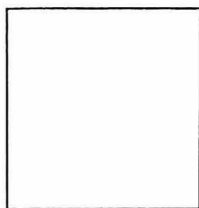
- Trong ví dụ trên chúng ta thấy cả 2 nhân vật rùa đều có thể được điều khiển di chuyển độc lập nhưng cùng dùng chung các sự kiện gọi hàm như onclick() hay onscreenclick(). Do vậy không thể điều khiển 2 rùa này hoàn toàn độc lập.

- Muốn điều khiển 2 hay nhiều hơn con rùa một cách độc lập hoàn toàn cần sử dụng các công cụ khác mạnh hơn. Các kiến thức đó nằm ngoài phạm vi cuốn sách này.

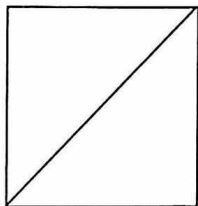


## Câu hỏi - bài tập

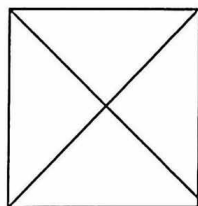
1. Viết chương trình cho rùa vẽ các hình sau:



(a)

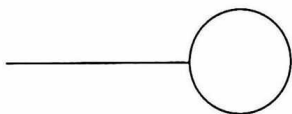


(b)

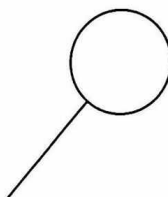


(c)

2. Viết chương trình vẽ các hình sau:

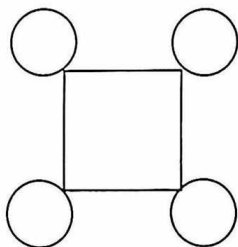


(a)

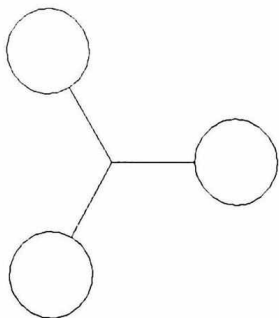


(b)

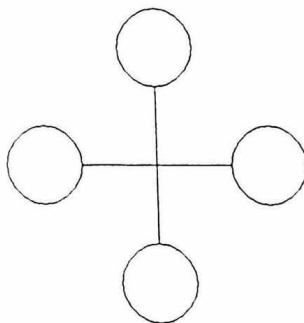
3. Viết chương trình vẽ hình sau:



4. Viết chương trình vẽ các hình sau:



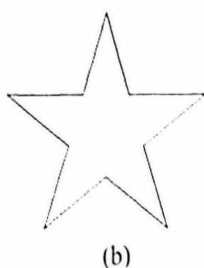
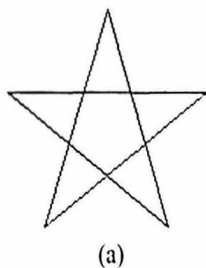
(a)



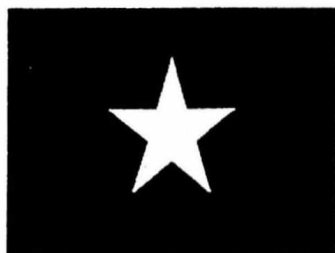
(b)



5. Tổng quát bài tập 4, vẽ hình bông hoa  $n$  cánh, mỗi cánh hoa là hình tròn.
6. Viết chương trình vẽ các hình ngôi sao sau đây.



7. Viết chương trình vẽ quốc kỳ Việt Nam, các thông số cụ thể như sau.



Một số thông số của lá cờ:

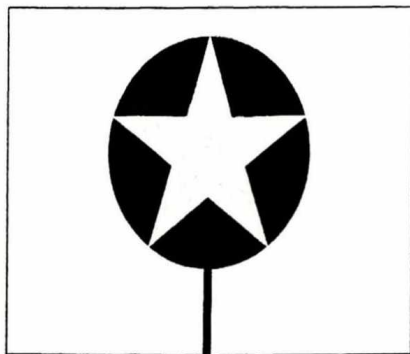
- Chiều rộng 200, chiều dài 300.
- Khoảng cách từ tâm đến 1 đỉnh của ngôi sao: 60.

Hay tổng quát:

- Chiều dài lá cờ dài gấp  $3/2$  chiều rộng.
- Khoảng cách từ tâm đến đỉnh ngôi sao bằng  $1/5$  chiều dài.

8. Viết chương trình vẽ đèn ông sao năm cánh như hình sau.

Khoảng cách từ tâm đến đỉnh ngôi sao là 120 point.

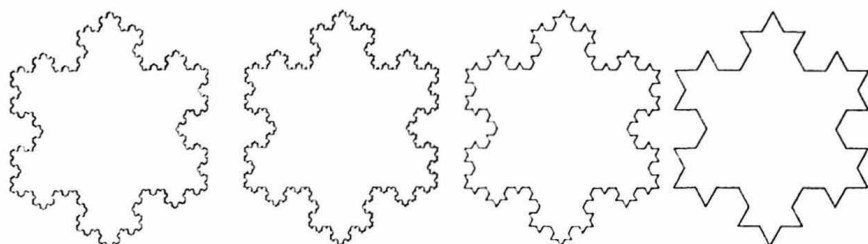


Yêu cầu bắt buộc:

- Hình tròn màu đỏ.
- Ngôi sao màu vàng.
- Cản đèn ông sao màu xanh lá cây.

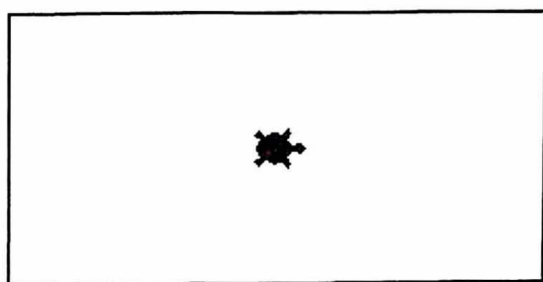
Chương trình cần viết đủ tổng quát để có thể thay đổi vị trí tâm và bán kính hình tròn.

9. Thiết kế chương trình chỉ cần 1 thủ tục có thể vẽ được các đường cong Koch với độ phức tạp khác nhau như các hình dưới đây.



Yêu cầu viết 1 thủ tục có dạng Koch-Curve với tham số là **level** - số mức của đường cong này. Tương ứng hình trên là các mức level = 5, 4, 3, 2 của thủ tục này.

10. Viết chương trình mô tả trò chơi tương tác đơn giản sau:



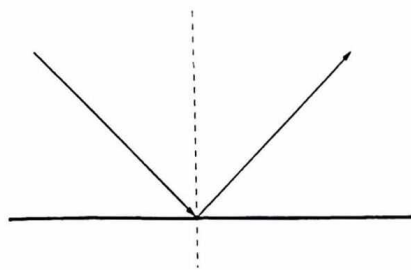
Vị trí ban đầu của con rùa nằm ở tâm hình chữ nhật.

Trên màn hình ban đầu có một hình chữ nhật và con rùa ở trung tâm như hình trên. Trò chơi bắt đầu khi người chơi nhấp chuột lên màn hình.

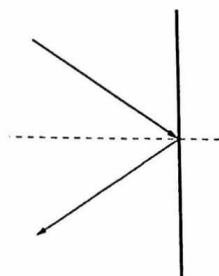
- Khi nhấp chuột lên màn hình, chuột sẽ quay hướng về con rùa chuột và chuyển động về hướng vị trí vừa nhấp chuột với vận tốc không đổi tỷ lệ với khoảng cách từ rùa đến vị trí nhấp chuột.

- Khi đã chuyển động, rùa sẽ di chuyển với vận tốc không đổi, nếu gặp cạnh của hình chữ nhật thì bật lại theo nguyên tắc đối xứng gương.

Nguyên tắc đối xứng gương được mô tả như trong các hình sau.



Rùa di chuyển và chạm cạnh bên dưới của hình chữ nhật.



Rùa di chuyển và chạm cạnh bên phải của hình chữ nhật.

## Chương 16. Bắt lỗi và kiểm soát lỗi chương trình

### Mục tiêu:

- Nhận dạng lỗi khi viết và chạy chương trình Python.
- Một vài cách bắt lỗi và kiểm soát lỗi khi viết chương trình.



### Nội dung chính

Một trong những nhiệm vụ quan trọng của người viết chương trình là làm sao để ít lỗi xảy ra nhất. Có cách nào để tìm được lỗi và sửa lỗi trong khi viết chương trình được hay không, đó là vấn đề đặt ra cho mọi chuyên gia lập trình.

Trước hết chúng ta phân biệt 2 loại lỗi thường gặp phải khi lập trình.

#### 1. Lỗi syntax và lỗi logic nội tại

Lỗi **syntax** là lỗi khi viết sai cú pháp lệnh của ngôn ngữ lập trình. Khi gặp lỗi này Python lập tức dừng chương trình và đưa ra thông báo lỗi dạng **SyntaxError**.

Một vài ví dụ lỗi ngữ pháp này.

```
>> while True print("Hello World")
SyntaxError: invalid syntax
>>> iff 4 < 5:
SyntaxError: invalid syntax
>>> 4 => 4
SyntaxError: invalid syntax
>>> abc'
SyntaxError: EOL while scanning string literal
>>>
```

Lỗi **syntax** phát sinh khi chúng ta viết sai các qui định cơ bản của ngôn ngữ lập trình. Với Python, các lỗi syntax đều phát sinh mã lỗi **SyntaxError**. Với các lỗi này rất dễ sửa vì có thể nhìn thấy ngay trên dòng lệnh.

Loại lỗi thứ 2 khó phát hiện hơn nhiều là lỗi phát sinh không phải cho cách viết sai cú pháp, mà lỗi thường phát sinh trong quá trình chạy chương trình, các lỗi này phát sinh do nội tại logic của chương trình tạo ra. Chúng ta sẽ gọi loại lỗi thứ 2 này là các lỗi **logic**. Các lỗi này rất khó phát hiện và sửa lỗi.

Ví dụ các lỗi loại này:

- Lỗi khi nhập liệu: Người dùng nhập dữ liệu không chính xác.
- Lỗi chia cho 0.
- Lỗi chỉ số vượt quá giới hạn của dãy.
- Lỗi thời gian chạy quá lâu.
- Lỗi lời gọi hàm có tham số không đúng kiểu.

Với các lỗi kiểu logic này, Python có một cơ chế cho phép bắt lỗi, kiểm tra lỗi khá hiệu quả để người viết chương trình chủ động sửa lỗi hoặc tìm cách tránh lỗi.

Trong bài học này chúng ta chủ yếu sẽ tìm hiểu các loại lỗi dạng này. Cơ chế tìm lỗi này có tên chung là Ngoại lệ - Exception.

## 2. Lỗi Exception (ngoại lệ)

Đặc điểm chung của các lỗi Exception như sau:

- Tất cả các lỗi loại này đều được Python đặt tên và đánh mã lỗi, mã lỗi này được gọi là các giá trị **Exception**. Mỗi mã lỗi Exception có một tên riêng và mô tả lỗi tương ứng. Ví dụ mã lỗi `IndexError` là lỗi liên quan đến mô tả chỉ số vượt quá giới hạn của kiểu dữ liệu tuần tự. Bảng các mã lỗi Exception thường gặp nằm ở cuối bài học này.

- Việc bắt lỗi sẽ được thiết lập bởi một cơ chế đặc biệt thông qua lệnh **try ... except ...**. Chúng ta cùng quan sát ví dụ đơn giản sau.

```
1. while True:
2. try:
3. x = int(input("Hãy nhập một số: "))
4. break
5. except ValueError:
6. print("Sai rồi, bạn cần nhập một số. Hãy nhập lại.")
```

Giả sử lệnh nhập dữ liệu (dòng 3) cần bắt lỗi vì người dùng hay nhập sai. Mục đích của chúng ta là nếu người dùng nhập lỗi sẽ thông báo để người dùng biết và nhập dữ liệu mà không bị văng ra khỏi chương trình.

Cách “bắt lỗi” sẽ được thể hiện như chương trình trên. Lệnh cần bắt lỗi được đưa vào cặp lệnh: **try ... except ...** và tất cả đưa vào vòng lặp **while**. Khi thực hiện lệnh trên, nếu người dùng nhập dữ liệu sai, tức là khi mã lỗi là **ValueError** thì lệnh **except** sẽ bắt lỗi và lệnh **print()** sau **except** sẽ thực hiện thông báo cho người dùng biết và vòng lặp thực hiện lại. Nếu người dùng nhập đúng, không có lỗi, lệnh ngắt **break** thực hiện sẽ kết thúc lệnh nhập dữ liệu trên.

## 3. Bắt lỗi Exception với lệnh try, mức đơn giản

Bắt lỗi Exception với lệnh try theo các bước như sau:

1. Xác định lệnh, nhóm lệnh cần và có thể gây lỗi, cần kiểm soát lỗi.
2. Xác định các kiểu lỗi Exception có thể xảy ra với các lệnh này.
3. Xác định các tương tác với lỗi nếu lỗi xảy ra, ví dụ thông báo cho người dùng, tìm cách đi vòng tránh lỗi, dùng lệnh khác thay thế, ....
4. Dùng nhóm lệnh **try ... except ...** để thực hiện các công việc trên.

Cấu trúc bắt lỗi Exception với lệnh try, mức đơn giản như sau.

**Cách 1:** Với mỗi lỗi thực hiện các lệnh riêng để xử lý lỗi.

---

**try:**

<lệnh hoặc nhóm lệnh cần bắt lỗi>

**except <Mã lỗi Exception 1>:**

<các lệnh xử lý 1>

**except <Mã lỗi Exception 2>:**

<các lệnh xử lý 2>

.

.

.

**except <Mã lỗi Exception N>:**

<các lệnh xử lý N>

---

Cách thực hiện nhóm lệnh trên như sau:

- Chương trình thực hiện nhóm lệnh <nhóm lệnh cần bắt lỗi> đứng sau lệnh try.
- Nếu không lỗi thì chuyển sang lệnh tiếp theo sau lệnh try.
- Nếu có lỗi, và lỗi này trùng với một trong các <mã lỗi Exception> có trong các lệnh except thì sẽ thực hiện <các lệnh xử lý> tương ứng.
- Nếu có lỗi và lỗi này không trùng với một trong các <mã lỗi Exception> đã ghi ở trên thì dừng chương trình, thông báo lỗi ra cửa sổ I/O.

**Cách 2:** Gộp các lỗi và xử lý chung bởi một lệnh.

---

**try:**

<lệnh hoặc nhóm lệnh cần bắt lỗi>

**except (<Exception1>, <Exception2>, ..., <ExceptionN>):**

<các lệnh xử lý chung cho tất cả các lỗi>

---

Cách thực hiện như sau:

- Chương trình thực hiện nhóm lệnh <nhóm lệnh cần bắt lỗi> đứng sau lệnh try.
- Nếu không lỗi thì chuyển sang lệnh tiếp theo sau lệnh try.
- Nếu có lỗi, và lỗi này trùng với một trong các <mã lỗi Exception> có trong nhóm mã lỗi liệt kê sau từ khóa except thì sẽ thực hiện <các lệnh xử lý chung> tương ứng.
- Nếu có lỗi và lỗi này không trùng với một trong các <mã lỗi Exception> đã ghi ở sau từ khóa except thì dừng chương trình, thông báo lỗi ra cửa sổ I/O.

Như vậy cách bắt lỗi trên vẫn chưa đầy đủ vì có thể chưa bắt hết được các kiểu lỗi. Cách giải quyết này vì vậy được coi là ở mức đơn giản.

Chúng ta cùng xem một ví dụ minh họa cho cách bắt lỗi này.



## Exception-1.py

---

```
1. import random
2. for i in range(5): # Loop 5 times
3. print("Bắt đầu vòng lặp:", i)
4. try:
5. r = random.randint(1, 3) # r ngẫu nhiên 1, 2, 3
6. if r == 1:
7. print(int('Fred')) # Chuyển số nguyên sai
8. elif r == 2:
9. [][2] = 5 # Gán index không tồn tại
10. else:
11. print(3/0) # Chia cho số 0
12. except ValueError:
13. print("Không thể chuyển thành số nguyên")
14. except IndexError:
15. print("Chỉ số vượt quá giới hạn")
16. except ZeroDivisionError:
17. print("Chia cho số 0")
18. print("Kết thúc vòng lặp:", i)
```

---

Kết quả thực hiện chương trình trên có thể như sau.

---

```
Kết thúc vòng lặp: 0
Bắt đầu vòng lặp: 1
Chỉ số vượt quá giới hạn
Kết thúc vòng lặp: 1
Bắt đầu vòng lặp: 2
Không thể chuyển thành số nguyên
Kết thúc vòng lặp: 2
Bắt đầu vòng lặp: 3
Chỉ số vượt quá giới hạn
Kết thúc vòng lặp: 3
Bắt đầu vòng lặp: 4
Không thể chuyển thành số nguyên
Kết thúc vòng lặp: 4
```

---

## 4. Bắt lỗi Exception, mức đầy đủ

Cách bắt lỗi trên, như chúng ta thấy, là vẫn chưa đầy đủ và an toàn. Chương trình vẫn có thể gặp lỗi và dừng đột ngột khi đang chạy. Để khắc phục lỗi này chúng ta sẽ chuyển sang cách bắt lỗi đầy đủ hơn, đảm bảo chương trình không bị dừng lại trong quá trình bắt lỗi.

Lệnh **try ... except ...** trong trường hợp này sẽ có thêm một lựa chọn bắt lỗi ngoại lệ cho tất cả các lỗi Exception chưa được nhắc đến. Cách viết này rất đơn giản, là một mở rộng nhanh của cách bắt lỗi đã nêu trên.

Sau đây là cú pháp của lệnh bắt lỗi mức đầy đủ.

---

```

try:
 <lệnh hoặc nhóm lệnh cần bắt lỗi>
except <Mã lỗi Exception 1>:
 <các lệnh xử lý 1>
except <Mã lỗi Exception 2>:
 <các lệnh xử lý 2>
.
.
.
except <Mã lỗi Exception N>:
 <các lệnh xử lý N>
except Exception:
 <các lệnh xử lý lỗi khác>

```

---

hoặc theo cách 2, cú pháp của lệnh đầy đủ như sau:

---

```

try:
 <lệnh hoặc nhóm lệnh cần bắt lỗi>
except (<Exception1>, <Exception2>, ..., <ExceptionN>):
 <các lệnh xử lý chung cho tất cả các lỗi>
except Exception:
 <các lệnh xử lý khác>

```

---

Ví dụ đã nêu trong mục trên sẽ được viết mở rộng cho phương án bắt lỗi đầy đủ như sau.

### Exception-2.py

---

```

1. import random
2. for i in range(10):
3. try:
4. r = random.randint(1, 4) # r ngẫu nhiên 1, 2, 3, 4
5. if r == 1:
6. print(int('Fred')) # Chuyển đổi số không đúng
7. elif r == 2:
8. [] [2] = 5 # Lỗi dùng index không tồn tại
9. elif r == 3:
10. print({}[1]) # Lỗi dùng key không tồn tại
11. else:
12. print(3/0) # Lỗi chia cho 0
13. except ValueError:
14. print("Không thể chuyển thành số nguyên")
15. except IndexError:
16. print("Chỉ số ngoài giới hạn")
17. except ZeroDivisionError:
18. print("Chia cho số 0")
19. except Exception: # Bắt các lỗi Exception còn lại
20. print("Lỗi khác")

```

---

## 5. Bắt lỗi chi tiết

### 5.1. Thông tin chi tiết lỗi

Với mỗi mã lỗi Exception, Python cung cấp các thông tin chi tiết cho từng lỗi này. Chương trình có thể bắt lỗi và thông báo chi tiết các lỗi này. Cách làm đơn giản được trình bày trong phần tiếp theo.

Quan sát ví dụ sau:

---

```
while True:
 try:
 x = int(input("Hãy nhập một số: "))
 break
 except ValueError as Err:
 print("Sai rồi, bạn đã nhập sai: ", Err)
 print("Hãy nhập lại.")
print("Bạn đã nhập đúng số: ", x)
```

---

Chỉ ra đối tượng lỗi Exception cụ thể là **Err** tại đây.

Trong chương trình trên, với câu lệnh sau, sự kiện bắt lỗi sẽ trả lại đối tượng <biến lỗi> dùng để mô tả chi tiết lỗi này.

---

**except** <Chỉ số lỗi> **as** <biến lỗi>

---

Sau đây là giao diện I/O của một ví dụ thực hiện chương trình trên. Thông tin chi tiết của lỗi được chỉ ra bởi đối tượng **Err**.

---

```
Hãy nhập một số: 1.56
Sai rồi, bạn đã nhập sai: invalid literal for int() with base
10: '1.56'
Hãy nhập lại.
Hãy nhập một số: 156
Bạn đã nhập đúng số: 156
```

---

### 5.2. Bỏ qua lỗi

Chúng ta có thể bỏ qua không thông báo lỗi hoặc thực hiện lệnh thông báo lỗi bằng cách dùng từ khóa **pass** trong phần lệnh **except** lỗi như sau.

---

```
while True:
 try:
 x = int(input("Hãy nhập một số: "))
 break
 except ValueError:
 pass
print("Bạn đã nhập đúng số: ", x)
```

---

Ví dụ đoạn chương trình trên có thể chạy như sau:

---

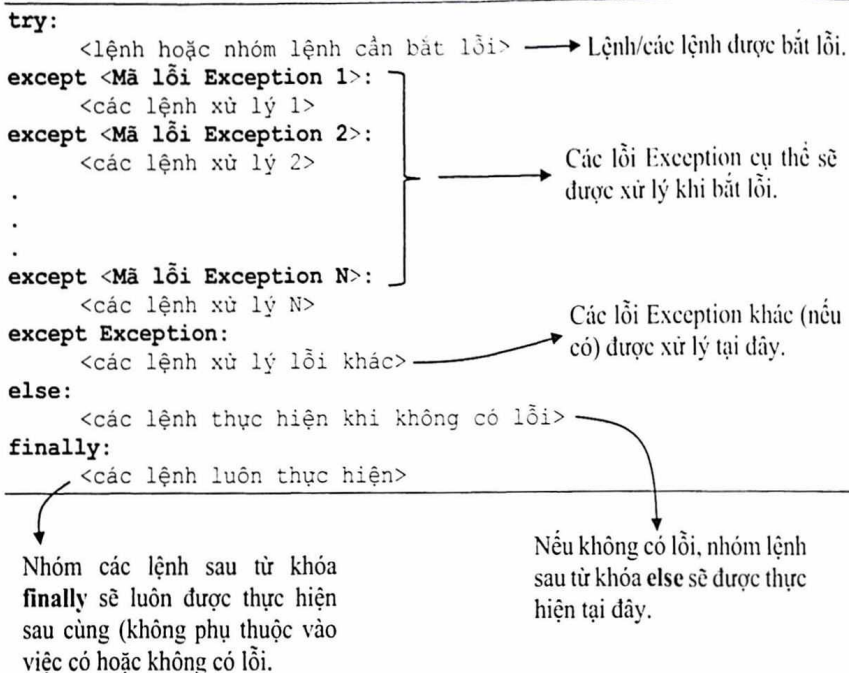
```
Hãy nhập một số: 1.45
Hãy nhập một số: abc
Hãy nhập một số: 123
Bạn đã nhập đúng số: 123
```

---

Chú ý: Bản thân từ khóa **pass** được dùng nhiều trong Python với ý nghĩa là một lệnh nhưng không thực hiện nhiệm vụ nào.

### 5.3. Nhóm lệnh *else* và *finally*

Cú pháp đầy đủ của lệnh bắt lỗi *try .... exception* có thêm 2 từ khóa *else* và *finally*. Ý nghĩa của 2 từ khóa này mô tả trong sơ đồ sau.



Sau đây là một ví dụ đơn giản cho cách sử dụng các từ khóa *else* và *finally*.

---

```
def divide(x, y):
 try:
 result = x / y
 except ZeroDivisionError:
 print("Phép chia cho 0!")
 else:
 print("Kết quả là:", result)
 finally:
 print("Đã thực hiện xong lệnh !")
```

---

Kết quả thực hiện hàm số trên, ví dụ, thể hiện như sau:

---

```
>>> divide(1,0)
Phép chia cho 0!
Đã thực hiện xong lệnh !
>>> divide(2,1.5)
Kết quả là: 1.3333333333333333
Đã thực hiện xong lệnh !
```

---

## 6. Một vài lỗi Exception có sẵn trong Python

Bảng các mã lỗi Exception thường gặp.

|                            |                                                      |
|----------------------------|------------------------------------------------------|
| <b>AttributeError</b>      | Lỗi liên quan đến truy cập thuộc tính của đối tượng. |
| <b>ZeroDivisionError</b>   | Lỗi phép chia cho 0.                                 |
| <b>OverflowError</b>       | Lỗi tràn số thập phân.                               |
| <b>EOFError</b>            | Lỗi khi cố đọc thông tin ở cuối file.                |
| <b>ModuleNotFoundError</b> | Lỗi không tìm thấy module.                           |
| <b>IndexError</b>          | Lỗi liên quan đến chỉ số của dãy.                    |
| <b>KeyError</b>            | Lỗi liên quan đến key của từ điển.                   |
| <b>NameError</b>           | Lỗi liên quan đến truy cập tên trong Namespace.      |
| <b>RecursionError</b>      | Lỗi khi vòng đệ quy quá giới hạn cho phép.           |
| <b>TypeError</b>           | Lỗi liên quan đến kiểu số.                           |
| <b>ValueError</b>          | Lỗi liên quan đến giá trị của đối tượng.             |
| <b>FileNotFoundError</b>   | Lỗi không tìm thấy file.                             |
| <b>FileExistsError</b>     | Lỗi khi ghi đè lên file đã tồn tại.                  |
| <b>IndentationError</b>    | Lỗi dòng lệnh thụt vào không đúng hàng.              |
| <b>SyntaxError</b>         | Lỗi cú pháp lệnh.                                    |

## 7. Tự tạo lỗi (raise Exception)

Trong phần trên chúng ta đã biết cách bắt lỗi Exception với các mã lỗi đã được thiết lập sẵn trong Python và lỗi được thiết lập trong chế độ bắt lỗi mặc định của Python.

Trên thực tế rất nhiều khi chúng ta muốn chương trình bắt các lỗi khác ngoài hệ thống bắt lỗi có sẵn của Python, ví dụ:

- Chúng ta muốn chương trình bắt lỗi khi người dùng nhập dữ liệu không thỏa mãn một điều kiện nào đó.
- Xây dựng một hàm số sẽ tự động tạo ra các lỗi Exception nếu tham số đầu vào không thỏa mãn điều kiện nào đó.

Giả sử chúng ta cần một hàm số dùng để nhập số tự nhiên từ bàn phím với điều kiện số nhập phải  $> 0$  và  $\leq 100$ . Nếu số đã nhập  $\leq 0$  hoặc  $> 100$  thì sẽ yêu cầu nhập lại.

Có thể thiết lập hàm số như sau:



---

```
def NhapTN():
 n = int(input("Nhập số tự nhiên: "))
 if n <= 0 or n > 100:
 print("Bạn nhập sai, hãy nhập lại. ")
 return n
```

---

Chương trình bắt lỗi Exception của hàm số trên được thực hiện tương tự bằng lệnh try như đã nêu trong các phần trên.

---

```
1. while True:
2. try:
3. x = NhapTN()
4. break
5. except ValueError:
6. print("Bạn đã nhập sai, nhập lại")
7. print("Bạn đã nhập đúng số:", x)
```

---

Tuy nhiên chương trình bắt lỗi trên chạy không đúng. Ví dụ một giao diện thực hiện như sau:

---

```
Nhập số tự nhiên: -10
Bạn nhập sai, hãy nhập lại.
Bạn đã nhập đúng số: -10
```

---

Lý do của việc không thể bắt lỗi của hàm số trên vì lỗi Exception không xuất hiện khi người dùng nhập số  $\leq 0$  hoặc  $> 100$ .

Như vậy vấn đề là làm thế nào để Python phát lỗi Exception khi người dùng nhập số  $\leq 0$  hoặc  $> 100$ . Python có lệnh **raise** cho phép tự tạo ra lỗi Exception. Một đáp án được mô tả trong chương trình sau. Python sẽ tạo ra lỗi **ValueError** khi người dùng nhập giá trị không phải nguyên hoặc nếu nguyên nhưng nằm ngoài vùng  $[1..100]$ .

---

```
8. def NhapTN():
9. n = int(input("Nhập số tự nhiên(1-100): "))
10. if n <= 0 or n > 100:
11. raise ValueError(n)
12. return n
13.
14. while True:
15. try:
16. x = NhapTN()
17. break
18. except ValueError:
19. print("Bạn đã nhập sai, nhập lại")
20. print("Bạn đã nhập đúng số:", x)
```

Tự tạo lỗi Exception bằng  
lệnh raise.

---

Ví dụ một giao diện thực hiện chương trình như sau.

---

```
Nhập số tự nhiên(1-100): 200
Bạn đã nhập sai, nhập lại
Nhập số tự nhiên(1-100): -10
Bạn đã nhập sai, nhập lại
Nhập số tự nhiên(1-100): asss
Bạn đã nhập sai, nhập lại
Nhập số tự nhiên(1-100): 46
Bạn đã nhập đúng số: 46
```

---

## 8. Mô hình đối tượng của lỗi Exception

### 8.1. Đối tượng Exception

Trong Python tất cả các mã lỗi dạng Exception đều là các đối tượng. Đối tượng gốc của các mã lỗi này là **BaseException**. Trong các đối tượng con của **BaseException** thì lớp con **Exception** là quan trọng nhất. Tất cả các mã lỗi thường gặp của Python đều là đối tượng con của đối tượng **Exception** này.

### 8.2. Tự định nghĩa Exception

Người sử dụng có thể tạo ra các lỗi Exception riêng của mình. Tất cả các đối tượng lỗi này đều phải là đối tượng con (hoặc cháu, chắt) của đối tượng Exception (xem mô hình cây các đối tượng Exception cơ bản của Python trong phần sau). Các mã lỗi tự định nghĩa được dùng trong các trường hợp riêng cụ thể của từng chương trình khi người lập trình không thể dùng các mã lỗi cơ bản hoặc muốn tạo ra các mã lỗi riêng biệt, đặc thù.

Ví dụ có thể tự định nghĩa một mã lỗi mới là *abcError* như sau, lớp này bắt buộc phải là lớp con của lớp Exception.

---

```
class abcError(Exception):
 def __init__(self,message):
 self.message = message
```

---

Quay lại ví dụ chương trình yêu cầu nhập một số tự nhiên trong khoảng từ 1 đến 100 đã được nêu trong mục trên. Trong ví dụ đó chúng ta dùng lệnh **raise** và tạo ra mã lỗi **ValueError** để xử lý. Trong chương trình dưới đây chúng ta sẽ định nghĩa một mã lỗi mới là **inputError** dùng để bắt lỗi riêng cho hàm nhập dữ liệu. Chương trình hoàn chỉnh như sau.

---

```
1. class inputError(Exception):
2. def __init__(self,message):
3. self.message = message
4. def NhapTN():
5. n = int(input("Nhập số tự nhiên: "))
6. if n <=0 or n > 100:
7. raise inputError("Bạn nhập không đúng theo yêu cầu.")
8. return n
9. while True:
10. try:
11. x = NhapTN()
12. break
13. except ValueError:
14. print("Bạn đã nhập sai, nhập lại")
15. except inputError as Err:
16. print(Err)
17. print("Bạn đã nhập đúng số:", x)
```

---

Kết quả chạy chương trình hoàn toàn chính xác như sau:

---

```
Nhập số tự nhiên: 120
Bạn nhập không đúng theo yêu cầu.
Nhập số tự nhiên: 99
Bạn đã nhập đúng số: 99
```

---

### 8.3. Mô hình cây các đối tượng Exception cơ bản của Python

Bảng các đối tượng và quan hệ đối tượng các Exception của Python.

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
 +-- StopIteration
 +-- StopAsyncIteration
 +-- ArithmeticError
 | +-- FloatingPointError
 | +-- OverflowError
 | +-- ZeroDivisionError
 +-- AssertionError
 +-- AttributeError
 +-- BufferError
 +-- EOFError
 +-- ImportError
 | +-- ModuleNotFoundError
 +-- LookupError
 | +-- IndexError
 | +-- KeyError
 +-- MemoryError
 +-- NameError
 | +-- UnboundLocalError
 +-- OSError
 | +-- BlockingIOError
 | +-- ChildProcessError
 | +-- ConnectionError
 | | +-- BrokenPipeError
 | | +-- ConnectionAbortedError
 | | +-- ConnectionRefusedError
 | | +-- ConnectionResetError
 | +-- FileExistsError
 | +-- FileNotFoundError
 | +-- InterruptedError
 | +-- IsADirectoryError
 | +-- NotADirectoryError
 | +-- PermissionError
 | +-- ProcessLookupError
 | +-- TimeoutError
 +-- ReferenceError
 +-- RuntimeError
 | +-- NotImplementedError
 | +-- RecursionError
 +-- SyntaxError
 | +-- IndentationError
 | +-- TabError
 +-- SystemError
 +-- TypeError
 +-- ValueError
 | +-- UnicodeError
 | | +-- UnicodeDecodeError
 | | +-- UnicodeEncodeError
 | | +-- UnicodeTranslateError
```



## Câu hỏi - bài tập

1. Viết đoạn chương trình yêu cầu nhập một số nguyên từ bàn phím, trong đó sử dụng bẫy lỗi (Exception), nếu người dùng nhập số thực hoặc văn bản. Nếu nhập sai, chương trình sẽ thông báo liên tục “Bạn cần nhập số nguyên” cho đến khi nhập đúng thì thôi.

2. Mô tả ý nghĩa đoạn chương trình sau:

---

```
i = 0
Sum = 0
try:
 while True:
 Sum = Sum + lst[i]
 i = i + 1
except IndexError:
 pass
print("Tổng dãy số là =", Sum)
```

---

Viết lại đoạn chương trình trên không dùng bẫy lỗi Exception.

3. Mệnh đề nào dưới đây mô tả đúng ý nghĩa của bẫy lỗi Exception?

- a) Dùng bẫy lỗi để tìm lỗi logic trong chương trình.
- b) Dùng bẫy lỗi để tìm lỗi syntax trong chương trình.
- c) Dùng bẫy lỗi để nhanh chóng tìm ra lỗi khi người dùng nhập dữ liệu.
- d) Dùng bẫy lỗi để tránh cho chương trình bị dừng lại nếu gặp lỗi trong khi đang chạy.

4. Dùng một lệnh `try .... exception` có thể bẫy được bao nhiêu loại lỗi:

- a) 1
- b) 2
- c) Không hạn chế.
- d) Tùy vào từng trường hợp cụ thể.

5. Chương trình sau có thể gây lỗi logic như thế nào:

---

```
Sum = 0
f = open('data.dat', 'r')
for line in f:
 Sum = Sum + int(line)
f.close()
print(sum)
```

---

Muốn bắt lỗi trong chương trình trên cần sửa như thế nào?

6. Đoạn lệnh sau sẽ báo lỗi loại gì?

---

```
while True print(Hôm nay là ngày đẹp trời)
```

---

- a) Lỗi Exception.
- b) Lỗi Syntax.

- c) Lỗi Logic.
- d) Không báo lỗi.

7. Hãy sửa và thêm các lệnh bắt lỗi vào đoạn chương trình sau.

---

```
def Change(A, p):
 for i in range(p):
 A[i] = 0
```

---

8. Giả sử chúng ta đã định nghĩa một hàm  $f(x)$ , sau đó thực hiện lời gọi hàm, ví dụ như sau:

---

```
1. def f(x):
2. <thân hàm số>
3. return <giá trị hàm số>
4.
5. a = f(10)
```

---

Để kiểm soát lỗi tại dòng 5, chúng ta sẽ dùng mã lỗi Exception nào?

- a) ValueError.
- b) TypeError.
- c) SyntaxError.
- d) NameError.

9. Các lỗi Exception trong Python thuộc kiểu dữ liệu gì?

- a) Tuple
- b) List
- c) Là các lớp con của lớp Exception
- d) Không thuộc kiểu dữ liệu nào.

10. Các mã lỗi Exception do người dùng tự định nghĩa cần phải thỏa mãn điều kiện gì?

- a) Cần có tên kết thúc bằng cụm từ Error.
- b) Là lớp con của lớp Exception.
- c) Là lớp con của lớp BaseException.

11. Khi dùng lệnh **raise** để tự tạo lỗi, có thể dùng mã lỗi thông thường hay không?

- a) Có thể được.
- b) Không thể, bắt buộc phải tự định nghĩa một mã lỗi riêng.

12. Hàm số `GenList(n)` có chức năng yêu cầu người dùng nhập và trả về danh sách (list) có  $n$  phần tử là số tự nhiên. Viết hàm số này với yêu cầu bắt lỗi Exception chính xác.



# INDEX

## ASCII

bảng mã chuẩn, 120

## Bảng mã

ASCII, 121

## Biến nhớ, 26

Quy tắc đặt tên, 30

## Biểu thức, 27

## bool

kiểu dữ liệu, 163

toán tử so sánh, 166

## Dãy

Fibonacci, 182

## Dãy Escape, 124

## Dãy số

Fibonacci, 83

nguyên tố, 84

## Độ quy, 180

## định danh, 30

## Dữ liệu có cấu trúc, 92

## Exception

Bắt lỗi, 239

Các Exception thường gặp, 245

Lỗi dạng, 239

Mô hình cây, 248

Tự định nghĩa, 247

Tự tạo lỗi, 245

## files

đọc thông tin từ file, 150

Đọc và ghi, 148

Ghi thông tin, 153

open(), 148

## generator

khởi tạo bằng hàm với yield, 104

tạo nhanh, 103

## Ghi chú, 23

## Hàm

divmod, 32

eval(), 42

float(), 40

input, 34

int(), 40

str(), 41

tell(), 150

## Hàm Lambda, 58

## Hàm số

chr(), 123

count(), 128

Định nghĩa, 51

find(), 128

hàm có giá trị, 53

hàm countdown, 183

hàm lambda, 58

hàm tính giai thừa, 183

hàm void, 52

hàm xác định số nguyên tố, 58

kiểu truyền giá trị, 53

ord(), 123

phạm vi áp dụng, 54

range(), 47, 81

replace(), 129

Tham số lựa chọn, 56

Tham số thực sự, 56

identifier, 30

## If

lệnh if, 168

## Input, 39

## Keyword, 32

## Kiểu bool, 43

## Kiểu dữ liệu, 26

cơ bản, 26

file, 149

list, 75

logic, 43

tuần tự, 79

Tuple, 82

## Kiểu dữ liệu tuple, 82

## Lệnh

\_\_init\_\_, 69

break, 171

continue, 172

dir(), 67

for, 46

gán, 30

gán đồng thời, 30

gán thay đổi, 31

help, 22

if, 44

raise, 246

readline(), 151

readlines(), 152

type(), 26

viết lệnh trên nhiều dòng, 33

Viết lệnh trên nhiều dòng, 23

while, 47

- write(), 154
- writelines(), 156
- Lệnh for, 79
- Lệnh import, 110
- List
  - các phép toán, 76
  - các phương thức, 77
  - chỉ số âm, 96
  - chỉ số của list, 80
  - lệnh for trên list, 79
  - lệnh for với list, 94
  - lệnh gán đồng thời, 93
  - Liên kết tham biến, 98
  - Liên kết tham chiếu list, 100
  - list comprehension, 102
  - list của list, 89
  - List của List, 90
  - phép +, 76
  - slicing, 96
- Lỗi
  - bất lỗi chi tiết, 243
  - lỗi syntax, 238
  - phân biệt lỗi, 238
- Lớp con, 71
- Ma trận
  - mô tả, 91
  - nhập, 91
- Module
  - cmath, 111
  - math, 111
  - random, 114
  - Tự thiết lập, 113
- Namespace
  - Khái niệm, 34
- Ngẫu nhiên
  - hàm random(), 114
  - Sinh giá trị ngẫu nhiên từ dãy, 115
  - Sinh số ngẫu nhiên, 114
- Nguyên tắc chuyển mạch nhanh logic, 173
- Object
  - Khởi tạo, 64
  - Phương thức, 65
  - Thuộc tính, 65
  - Tổng quan về OOP, 63
- Phép tính lũy thừa, 33
- PyCharm, 20
- PyScripter, 21
- Python
  - Cài đặt, 15
  - IDE, 16
  - Lịch sử ngôn ngữ, 15
  - Shell, 15
  - Trợ giúp, 22
  - Từ khóa, 32
  - Số phức, 111
  - String
    - f-string, 142
    - Kiểu dữ liệu, 119
    - Lệnh split và join, 130
    - phương thức căn chỉnh, 126
    - tạo khuôn bằng format(), 132
    - tự điều chỉnh khoảng trống thừa, 127
  - Tập hợp
    - các phép toán, 209
    - kiểu dữ liệu, 205
    - Tạo tập hợp, 206
    - Thêm, bớt phần tử, 208
  - Toán tử
    - chồng, 69
    - logic, 44
    - so sánh, 44
  - Toán tử in, 173
  - Từ điển
    - Các phương thức, 195
    - kiểu dữ liệu, 194
  - Từ khóa, 32
  - Tuần tự
    - kiểu dữ liệu, 102
  - Tuple
    - Các phương thức, 83
    - Kiểu dữ liệu, 82
    - phép cộng, 83
  - Turtle
    - các lệnh điều khiển, 218
    - hàm sự kiện, 227
    - hướng, 217
    - môi trường đồ họa, 216
    - thiết lập màu vẽ, 223
    - thiết lập tương tác, 225
    - Tọa độ, 217
  - Vòng lặp lồng nhau, 89
  - while
    - lệnh lặp while, 170
  - Wingware Personal, 18

## Tài liệu tham khảo

- [1]. Tài liệu gốc của Python. <https://www.python.org/doc/>
- [2]. Fundamental of Python Programming. Richard L. Halterman. 2019.
- [3]. Invent Your Own Computer Games with Python. 2nd Edition. Al Sweigart.
- [4]. Python® Notes for Professionals. <https://goalkicker.com/PythonBook/>
- [5]. LOGO Programming Book.
- [6]. Introduction to Algorithms. Third Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
- [7]. Discrete Mathematics and Its Applications. 7th Edition. Kenneth H. Rosen.
- [8]. Thiết kế phần mềm và trò chơi giáo dục. Bùi Việt Hà. NXB Đại học quốc gia Hà Nội. 2018.
- [9]. Thiết kế trò chơi Lines. Bùi Việt Hà.
- [10]. STEM Computing. Bùi Việt Hà.
- [11]. Các bản tin CS4S 2017, 2018, 2019.
- [12]. Introduction to The Design and Analysis of Algorithm. 3rd Edition. Anany Levitin.
- [13]. Computing in the national curriculum. A guide for primary teachers.
- [14]. Computing in the national curriculum. A guide for secondary teachers.
- [15]. Learn to Program with Scratch. Majed Marji.
- [16]. Scratch 2.0 Game Development. Sergio van Pul, Jessica Chiang.
- [17]. Computer Science Concepts in Scratch. Michal Armoni and Moti BenAri, 2013. <http://stwww.weizmann.ac.il/g-cs/scratch/scratch-14-textbook-1-0-one-side.pdf>
- [18]. Program or be Programmed: Ten Commands for a Digital Age. Rushkoff, D. OR Books, 2009.
- [19]. CS unplugged. Giảng dạy khoa học máy tính không cần máy tính. <http://www.csunplugged.org>
- [20]. Chương trình, chuẩn kiến thức môn Tin học. NXB Giáo dục, Hà Nội, 2000.
- [21]. Chương trình giáo dục phổ thông tổng thể trong chương trình giáo dục phổ thông mới. Tài liệu chính thức Bộ Giáo dục và Đào tạo, 2017.
- [22]. Computing in the national curriculum A guide for primary teachers. Tài liệu dành cho GV cấp Tiểu học Anh quốc dạy môn Tin học. <http://www.computingschool.org.uk/data/uploads/CASPrimaryComputing.pdf>
- [23]. Shut Down or Restart. <https://royalsociety.org/topics-policy/projects/computing-in-schools/report/>  
<https://royalsociety.org/~media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf>
- [24]. Computing at School, The Raspberry Pi Education Manual (CAS, 2012). [http://pi.cs.man.ac.uk/download/Raspberry\\_Pi\\_Education\\_Manual.pdf](http://pi.cs.man.ac.uk/download/Raspberry_Pi_Education_Manual.pdf)

- [25]. Scratch Programming For Teens. Jerry Lee Ford, Jr.
- [26]. Super Scratch Programming Adventure! The LEAD Project
- [27]. Computational thinking. A guide for teachers. Andrew Csizmadia, Simon Humphreys, National Coordinator, Computing At School.
- [28]. Computer Science: A curriculum for schools.  
<http://www.computingschool.org.uk/>
- [29]. K12-Computer Science Framework.  
<https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf>.
- [30]. Lịch sử phát triển môn Tin học ở Việt Nam. CS4S số 3 (2017).
- [31]. Đề cương khung chương trình môn Tin học của Anh quốc (bản mới). CS4S số 1 (2017).
- [32]. Hiểu và đi trong cách mạng công nghiệp lần thứ tư. Hồ Tú Bảo. 2017.
- [33]. Tự học lập trình Scratch. Bùi Việt Hà. NXB Đại học quốc gia Hà Nội. 2018.
- [34]. Thiết kế phần mềm mô phỏng chơi Cờ Vua trên Scratch. Bùi Việt Hà.
- [35]. O-level Computing for teaching and learning. Curriculum Planning and Development Division.
- [36]. Kodu for Kids. QUE Publication.
- [37]. Oxford International Primary Computing. Volume 1-6. Oxford University Press.
- [38]. Góp ý cho đề cương môn Tin học trong Chương trình Giáo dục Phổ thông mới 2018. CS4S số 3 (2018).
- [39]. Chương trình khung (chính thức) môn Tin học trong Chương trình Giáo dục Phổ thông mới. Bộ Giáo dục và Đào tạo. 7-2018.
- [40]. Nhận xét, phân tích, góp ý cho Chương trình môn Tin học trong CTGDPT mới. Bùi Việt Hà. CS4S số 7 - 2018.
- [41]. Nhận xét, góp ý cho Chương trình môn Tin học trong CTGDPT mới. CS4S số 9 - 2018.
- [42]. Chương trình Giáo dục Phổ thông mới môn Tin học, phiên bản tháng 9-2018. Bộ Giáo dục và Đào tạo.
- [43]. A smarter way to learn Python. Mark Myers. San Bernardino, CA.
- [44]. Python Algorithms. Mastering Basic Algorithms in the Python Language. Magnus Lie Hetland. Apress. 2010.
- [45]. Data Structure and Algorithms with Python. Kent D.Lee, Steve Hubbard. Springer Publisher. 2015.

NHÀ XUẤT BẢN

**ĐẠI HỌC QUỐC GIA HÀ NỘI**

**16 Hàng Chuối - Hai Bà Trưng - Hà Nội**

Giám đốc – Tổng biên tập: (024)39715011

Hành chính: (024)39714899; Fax: (024)39724736

Quản lý xuất bản: (024) 39728806

Biên tập: (024) 39714896

**Chịu trách nhiệm xuất bản**

Giám đốc: TRẦN QUỐC BÌNH

**Chịu trách nhiệm nội dung**

Tổng biên tập: TS. NGUYỄN THỊ HỒNG NGÀ

Biên tập: ĐẶNG THỊ PHƯƠNG ANH

Chế bản: BÙI VIỆT HÀ

Trình bày bìa: NGUYỄN NGỌC ANH

Đối tác liên kết: CÔNG TY TNHH TIN HỌC VÀ NHÀ TRƯỜNG  
Nhà số 6 ngõ 518, phố Đội Cấn, phường Cống Vị, Q. Ba Đình, Hà Nội

**PYTHON CƠ BẢN**

Mã số: 1L-32ĐH2023

In 5.000 bản, khổ 19x27 tại Công ty Cổ phần in và Thương mại Ngọc Hưng  
Địa chỉ: Số 296 đường Phúc Diễn, tổ dân phố số 1, P. Xuân Phương, Q. Nam Từ Liêm, Hà Nội

Số xác nhận đăng ký xuất bản: 1889-2023/CXBIPH/09-170/ĐHQGHN, ngày 13/6/2023

Quyết định xuất bản số: 418 LK-TN/QĐ - NXB ĐHQGHN, ngày 13/06/2023

In xong và nộp lưu chiểu năm 2023.





# Tủ sách CS4S

Thời gian gần đây, cùng với phong trào STEM phát triển mạnh, các ngôn ngữ lập trình trực quan lần lượt ra đời như Scratch đã tạo ra một cuộc cách mạng mới, bùng nổ của giáo dục Tin học trong nhà trường. Lần đầu tiên trong lịch sử, định hướng Khoa học máy tính của môn Tin học được đưa xuống cho học sinh bậc Tiểu học. Cuộc cách mạng này sẽ có ý nghĩa to lớn và lâu dài mà chúng ta có lẽ chưa thể đánh giá hết ngày hôm nay. Rất may mắn là Python lại chính là ngôn ngữ lập trình thích hợp dùng trong nhà trường, cho lứa tuổi cấp THCS, THPT, phù hợp với cả 2 định hướng phát triển chính của môn Tin học hiện nay là STEM và ALGORITHM.

# PYTHON CƠ BẢN

ISBN: 978-604-324-096-2



Giá: 125.000đ



TR  
T